Automaattisen videoanalyysin sovellus jalkapalloon

Pelaajien liikkeiden seuraaminen videokuvasta

Mikko Kankainen

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Pro gradu -tutkielma
Toukokuu 2002

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Mikko Kankainen: Automaattisen videoanalyysin sovellus jalkapalloon. Pelaajien liikkeiden seuraaminen videokuvasta.
Pro gradu -tutkielma, 53 sivua.

Toukokuu 2002

Tässä tutkielmassa esitellään yleisesti hahmontunnistusta, mutta erityisesti keskitytään videoanalysointiin soveltuviin hahmontunnistustekniikoihin. Tutkielmassa kerrotaan myös kuvan suodattamisesta, eli miten kuvan tiettyjä piirteitä voidaan korostaa tai miten häiriöitä voidaan poistaa.

Tutkielman toisen puoliskon muodostavat eri hahmontunnistustekniikoiden testaaminen käytännössä ja näin saatujen tutkimustulosten esittely. Tutkimusongelmia ovat pelaajien automaattinen tunnistaminen, yksittäisten kuvien havaintojen yhdistäminen toisiinsa ja pelaajien todellisten sijaintien selvittäminen videokuvasta. Tutkielmassa tarkastellaan lisäksi käytössä olevan laitteiston rajoituksia ja sitä, miten algoritmeja voidaan optimoida, jotta ne toimisivat reaaliaikaisesti.

Automaattisella videoanalyysillä on käytännössä mahdotonta saada täydellistä peli-informaatiota. Analysointia vaikeuttavat tilanteet, joissa pelaajat menevät kameraan nähden limittäin. Hahmojen päällekkäisyyden käsittelyä voidaan helpottaa erilaisilla heuristiikoilla, esimerkiksi käyttämällä hyväksi seurattavien hahmojen väri-informaatiota. Hahmojen päällekkäisyyden lisäksi valaistuksen vaihtelut ja varjot aiheuttavat ongelmia hahmontunnistusalgoritmeille. Mikäli verrataan jalkapallovideon kahta peräkkäistä kuvaa, tunnistetaan vain liike. Jos videokuvaa verrataan tyhjästä kentästä otettuun kuvaan, häiritsee valaistuksen vaihtelu merkittävästi analyysiä. Analyysin tarkkuus laskee huomattavasti, kun kamera on noin puolen kentän päässä pelaajista. Yksi mahdollisuus analyysin tarkkuuden parantamiseen olisi käyttää kahta kentän eri päissä olevaa kameraa ja yhdistää näiden havainnot. Toinen mahdollisuus olisi resoluution kasvattaminen. Lisäksi videokuvan parillisten ja parittomien vaakarivien välisen vaihe-eron takia kuvan havaittua tarkkuutta voidaan parantaa käyttämällä vain joko parillisia tai parittomia vaakarivejä.

Avainsanat ja -sanonnat: hahmontunnistus, jalkapallo, videoanalyysi.

Sisällys

1. Johdanto

Tämän tutkimuksen aiheena on jalkapalloilijoiden liikkeiden automaattinen seuraaminen videokuvasta. Tätä tarkoitusta varten Tampereen yliopistossa on kehitetty VideoAnalyzer-ohjelma. Ohjelman avulla on päästy kokeilemaan erilaisia tekniikoita pelaajien ja heidän polkujensa tunnistamiseen. Polku tarkoittaa tässä tutkielmassa yhtenäistä tunnistettua reittiä, jonka pelaajan liikkeet kentällä muodostavat.

VideoAnalyzer sai alkunsa Tampereen yliopiston projektityönä, jonka Jyväskylän yliopiston Kilpa- ja huippu-urheilun kehittämiskeskus (KIHU) teetti vuonna 1999 tuottamaan syötettä kehittämälleen GameManager-ohjelmistolle. Projektiryhmän muodostivat projektipäälliköt Juha Vallittu ja Petri Vatanen sekä ryhmän jäsenet Markus Laine, Mika Kanerva, Markku Siermala ja Jari Välimäki. KIHUn päässä toimeksiantajana toimi Pekka Luhtanen ja yliopiston päässä Jyrki Nummenmaa. Itse tulin mukaan kesällä 2000 tietojenkäsittelytieteiden laitoksen harjoittelijana. Tehtävänäni oli jatkaa VideoAnalyzerin kehitystä. Kesällä keskityin pääasiassa algoritmien ja tietorakenteiden kehittämiseen. Syystalvella 2000 päätin kirjoittaa pro gradu -tutkielmani tästä aiheesta ja perehdyin alan kirjallisuuteen. Kirjoittamisen aloitin helmikuussa 2001.

Markku Siermala on kirjoittanut tutkimuksen jalkapallo-pelin liikeanalyysistä (katso Siermala [1999]). Tämän ja kesäharjoittelussa tekemäni kehitystyön pohjalta Markku Siermala, minä ja Jyrki Nummenmaa olemme kirjoittaneet laitosraportin ”Tracing footballers’ movements from video” (katso [Siermala *et al.*, 2000]). Pro gradu -tutkielmani perustuu näihin kahteen tutkimukseen. Nämä kolme muodostavat jatkumon ja sivuavat toisiaan jossain määrin myös tutkimustulosten osalta.

1.1. Motiivi

Kilpa- ja huippu-urheilun kehittämiskeskus on kehittänyt GameManager-ohjelmiston, jolla voidaan analysoida jalkapallo-otteluja. GameManageriin syötetään pelaajien liikkeet janoina metrin tarkkuudella ja erilaiset tapahtumat, kuten esimerkiksi syöttö tai maali. GameManager analysoi syötetyt tiedot ja tulostaa tilastoja peleistä. GameManagerilla videon analysointi tapahtuu siten, että analysoija katsoo jalkapallo-ottelua videolta ja arvioi silmämääräisesti, kuinka paljon kukin pelaaja liikkuu ja mitä tämä kentällä tekee. Menetelmä on luonnollisesti erittäin työläs ja hidas. Kokenut henkilö analysoi tunnissa noin viisi minuuttia peliaikaa [Luhtanen, 1999]. Tämä on ollut yksi este menetelmän laajemmalle leviämiselle.

Tarvitaan siis järjestelmä, joka analysoi jalkapallovideoita automaattisesti. Tällaisia järjestelmiä on kehitetty, mutta useimmat niistä ovat niin kalliita, etteivät ne ole

kovinkaan monen jalkapalloseuran ulottuvilla. Eräs järjestelmä esimerkiksi käyttää viittä kameraa pelin seurantaan. Lisäksi kameroiden sijoittelulle voi olla hankalia vaatimuksia. Käyttömahdollisuudet ovat melko rajalliset, jos kameran täytyy olla suoraan kentän yläpuolella.

1.2. Tavoite

Tavoitteena on ollut kehittää automaattinen jalkapallovideoiden analysointijärjestelmä, joka ei aseta käytettävälle laitteistolle liian kovia vaatimuksia. Analyysiin käytetään ainoastaan yhtä videokameraa. Kameran sijoittelulle on vaatimuksia, mutta ne eivät ole kovin ankaria. Koska jalkapallovideot vievät valtavasti tilaa kiintolevyltä, tarkoituksena on ollut pyrkiä reaaliaikaiseen järjestelmään, jolloin peli voidaan analysoida suoraan videolta tietokoneeseen liitettävän videodigitointikortin avulla. Jalkapallovideon analysointi käsin on aikaa vievää ja epätarkkaa, eli käyttäjän rooli analyysissä on saatava minimiin. Tähän päästään kehittämällä tehokkaita algoritmeja pelaajien ja polkujen automaattiseen tunnistamiseen.

1.3. Vaatimuksia jalkapallovideon analysoinnille

Tutkimuksessa (Siermala *et al.* [2000]) kävi ilmi, että koko kenttää ei saada mahtumaan videokuvaan. Koska tavoitteena oli kuitenkin yhden kameran systeemi, oli vain hyväksyttävä se seikka, että tietty kentän alue ei mahdu kuvaan (katso kuva 3.9). Tämän alueen minimoimiseksi kuvan on oltava mahdollisimman kattava. Kuvan pitää myös olla niin laaja, että siitä voidaan tunnistaa tiettyjä kentän kiintopisteitä, joita tarvitaan pelaajien todellisen sijainnin selvittämiseen. Toisaalta pelaajien pitää olla kuvassa niin isoja, että käyttäjä voi tunnistaa heidät, sillä automaattisen analyysin jälkeen käyttäjän apua tarvitaan nimenomaan pelaajien nimeämiseen. Mahdollisimman laajan kuvan ja pelaajien koon välille tulee löytää kompromissi. Kuvan mittasuhteisiin ei myöskään saisi syntyä vääristymistä.

Kamera kuvaa peliä koko ajan samasta kohdasta tarkentamatta yksittäisiä kohteita. Analyysissä käytetyt algoritmit edellyttävät, että kameran sijainti suhteessa kenttään tiedetään, samoin kuin kentän leveys ja pituus.

1.4. Tutkimusongelmia

Choi *et al.* [1997] ovat määrittäneet kolme automaattisen jalkapalloanalyysin pääongelmaa:

1. Kenttä pitäisi suodattaa pois, jotta pelaajat ja kentän kiintopisteet voitaisiin löytää. Koska tulosta käytetään kaikissa seuraavissa vaiheissa, sen olisi oltava mahdollisimman tarkka.

2. Kaikki pelaajat ja pallo pitäisi pystyä identifioimaan. Ongelmia syntyy, koska pelaajat liikuvat epäsäännöllisesti, hahmot törmäävät usein toisiinsa ja peittävät toisiaan.
3. Pelaajien absoluuttiset sijainnit pitäisi pystyä selvittämään.

Choi *et al.*:n systeemi eroaa sikäli tämän tutkimuksen systeemistä, että heidän tapauksessaan kamera liikkuu. Kiintopisteet pitää siis löytää kuvasta automaattisesti aivan kuten pelaajatkin. VideoAnalyzerin nykyisessä toteutuksessa käyttäjä määrittelee kentän kiintopisteet, joten tutkimusongelma ei ole aivan yhtä monimutkainen. Kentän suodattaminen on kuitenkin oleellinen tekijä ja siitä tekee vaikean valaistusolosuhteiden muutokset, varjot ja heijastukset. Edellämainitut seikat saavat helposti aikaan sen, että pelaajat sekottuvat taustaan.

Yleisesti tunnustettu suurin ongelma hahmontunnistuksessa on hahmojen päällekkäisyys. Choi *et al.* kiinnittävät huomiota tähän kohdassa kaksi. Koska jalkapalloilijat liikkuvat epäsäännöllisesti, liikkeiden ennustaminen on erittäin vaikeaa. Jos kaksi tai useampia hahmoja peittää toisensa, hahmontunnistusalgoritmit eivät yleensä pysty seuraamaan pelejä. Jalkapallo on erityisen hankala hahmontunnistuksen sovellusalue, sillä kunkin joukkueen pelaajilla on samanväriset peliasut. Siksi pelaajien asujen väri-informaatiota ei pystytä hyödyntämään päällekkäisyysongelmien ratkaisemiseen.

Pyrkimys reaaliaikaiseen järjestelmään asettaa rajoituksia sille, miten monimutkainen analyysivaihe voi olla. Analyysille saadaan ”lisäaikaa” sillä, että käsiteltävän kuvan kokoa pienennetään ja käsiteltävien kuvien määrää vähennetään. Kun kuvan resoluutio on pieni, etsittävät objektitkin ovat pieniä. Tämä vaikuttaa suoraan analyysin tarkkuuteen. Analyysin tarkkuus laskee huomattavasti, kun kamera on noin puolen kentän päässä pelaajista, sillä pelaajat ovat silloin kuvassa vain muutaman pikselin kokoisia.

Koko kentän kuvaaminen yhdellä kameralla aiheuttaa sen, että perspektiivistä tulee merkittävä tekijä. Pelaaja voi kaatua tai hypätä, jolloin seuranta-algoritmi erehtyy tulkitsemaan tilanteen niin, että pelaaja olisi liikkunut yhtäkkiä useita metrejä. Mitä kauempana kentällä tämä tapahtuu, sitä suuremmalta virhe näyttää. Perspektiivi aiheuttaa siis sen, että kaukana kentällä pelaajien liikkeet vaikuttavat suuremmilta suhteessa heidän kokoonsa kuin lähellä kameraa oltaessa.

1.5. Tutkielman sisällön esittely

Luvussa 2 esittelen videoanalysointiin soveltuvia tekniikoita. Ensiksi kerron hahmontunnistuksesta yleisesti. Alaluvussa 2.1 esittelen erilaisia videoanalysointiin

soveltuvia hahmontunnistustekniikoita. Alaluvussa 2.2 kerron kuvan suodattamisesta: miten kuvan tiettyjä piirteitä voidaan korostaa tai miten häiriöitä voidaan poistaa. Alaluvussa 2.3 kerron polkujen etsinnästä. Kun hahmot on ensiksi löydetty videosekvenssistä, havainnot hahmoista eri *ruuduissa* (frame) yhdistetään poluiksi. Luvussa 3 kerron omasta tutkimuksestani VideoAnalyzerin parissa. Ensiksi esittelen VideoAnalyzeriä, jonka jälkeen alaluvussa 3.1 kerron analyysin eri vaiheista. Ne ovat esiprosessointi, klusterointi ja polkujen etsintä. Alaluvussa 3.2 kerron VideoAnalyzerissä käytetyistä tietorakenteista ja muistinhallintaan liittyvistä ongelmista. Alaluvussa 3.3 kerron, kuinka objektien todelliset sijainnit kentällä lasketaan. Luvussa 4 kerron tutkimustuloksista. Alaluvussa 4.1 pohdin sitä, millaisia rajotteita reaaliaikaisuus asettaa analyysille. Alaluvussa 4.2 vertailen kokeiltuja tekniikoita. Hahmojen päällekkäisyyksien hallintaa pohdin alaluvussa 4.3 Alaluvussa 4.4 esitän vielä johtopäätöksiä ja mietteitä jatkotutkimukselle.

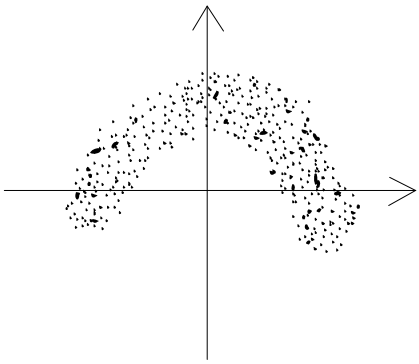
2. Videoanalysointiin soveltuvia tekniikoita

Kuvan *segmentointi* (segmenting) on olennainen osa monia konenäkösovelluksia. Se määritellään yleensä kuvan jakamiseksi eri alueisiin, jotka ovat homogeenisia joltakin / joiltakin ominaisuuksiltaan (esimerkiksi kirkkaus, väri, tekstuuri) [Jain *et al.*, 1999]. Jain *et al.* lainaavat Rosenfeldia ja Kakia, jotka kirjoittavat, että kuvan segmentointia voidaan pitää klusterointiongelmana.

Klusterointi (clustering) tarkoittaa *hahmojen* (patterns) automaattista luokittelua ryhmiin eli klustereihin. Klusterointi tapahtuu tunnistamalla samankaltaisuuksia. Jain *et al.* esittävät, miten Jain ja Dubes jaottelevat klusteroinnin viiteen osaan:

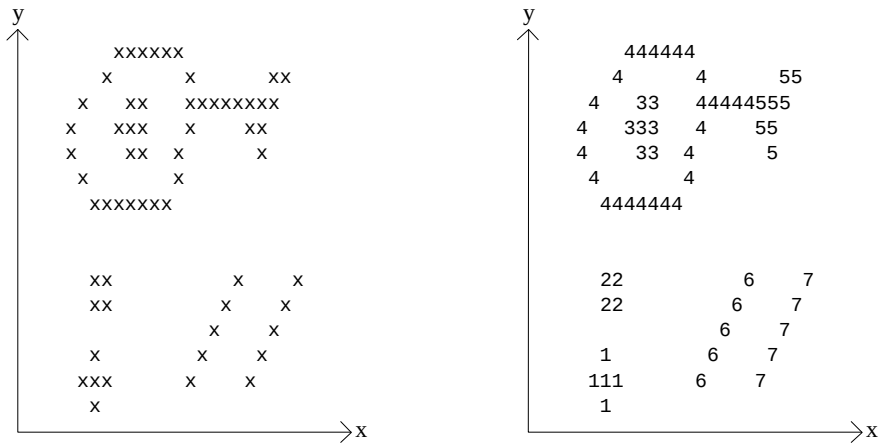
1. *Hahmon esittäminen* (pattern representation)
2. Hahmojen *samankaltaisuusluokitusten* (proximity measure) määrittäminen
3. Klusterointi tai ryhmittely
4. Datan abstrahointi (tarvittaessa)
5. Tulosteen arviointi (tarvittaessa)

Hahmon esittäminen tarkoittaa sitä, kuinka monta, minkä kokoisia ja tyyppisiä objekteja klusterointialgoritmin tulee löytää. Jain *et al.* [1999] antavat esimerkin hahmojen esittämisen ongelmallisuudesta. Kuvassa 2.1 on hahmo, jonka datapisteet ovat kaarevassa muodossa suurin piirtein samalla etäisyydellä origosta. Jos hahmojen esittämiseen käytetään karteesista koordinaatistoa, useat klusterointialgoritmit saattavat jakaa hahmon kahteen tai useampaan klusteriin, sillä se ei ole kompakti. Toisaalta jos hahmojen esittämiseen käytetään polaarikoordinaatistoa, löydetään todennäköisesti vain yksi klusteri.



Kuva 2.1. Hahmo, jonka datapisteet ovat kaarevassa muodossa. [Jain *et al.*, 1999]

Samankaltaisuusluokitusten määrittäminen tarkoittaa etäisyysfunktion määrittämistä, siis mitta-asteikon määrittämistä sille, mikä määrää objektien samankaltaisuuden. Alaluvun 2.1 hahmontunnistustekniikat ovat periaatteessa eri tapoja määritellä objektien samankaltaisuus. *Klusterointi tai ryhmittely* tarkoittaa havaintojen yhdistämistä. Jos kuvasta löydetään esimerkiksi kaksi soveliasta pikseliä, klusteroinnissa päätetään, kuuluvatko ne samaan objektiin. Klusterointia on havainnollistettu kuvassa 2.2. *Datan abstrahointi* tarkoittaa löydetyn klusterin kuvaamista. Kaikkea klusteriin liittyvää tietoa ei ole yleensä mielekästä tallentaa. Tavallisesti riittää, jos määritellään esimerkiksi klusterin keskipiste ja keskimääräinen väri tai kirkkaus. *Tulosteen arviointi* puolestaan tarkoittaa löydettyjen klustereiden tarkkuuden arviointia. Jos jalkapallokentältä löytyy 300 objektia (pelaajaa), klusteroinnissa lienee jotain vikaa.



(a) alkutilanne (b) klusteroinnin tulos

Kuva 2.2. Datan klusterointi. [Jain *et al.*, 1999]

Plan ja Marchantin [1997] mukaan ilmeinen tapa käsitellä käytännön tilanteita, joissa on useita objekteja, on jakaa kuva jollakin segmentointimenetelmällä. Kuva segmentoidaan joidenkin kriteerien tai jonkin menetelmän mukaan. Näin saatuja alueita voidaan ajatella objekteina. Tarkoituksena on ryhmitellä kriteerit täyttävät alueet järkeviksi kokonaisuuksiksi. Keskeistä ei ole se, kuvaavatko kokonaisuudet todellisia objekteja vai vain osaa taustasta. Pääasia on, että eri objektit eivät pääse sekoittumaan keskenään.

Voidaan ajatella, että toisessa ääripäässä on tilanne, jossa luokkiin kuulumisen kriteerit on määritelty ristiriidattomasti jo alusta alkaen, joten varsinainen objektien luokittelu on pelkkää mekaanista lajittelua. Toisessa ääripäässä on klusterointi. Silloin on kyseessä tilanne, jossa luokista ja luokiteltavista objekteista ei ole tietoa. Eli ennen kuin luokittelu voidaan tehdä, luokat pitää konstruoida objektien samankaltaisuuden ja erilaisuuden perusteella. [Watanabe, 1970]

Kovalevsky [1970] esittää hahmontunnistuksen formaalimmin. Hänen mukaansa kaikille hahmontunnistusongelmille on yhteistä joukko multidimensionaalisia signaaleja *v*, joiden perusteella tehdään päätöksiä. Päätös *d* tehdään yleensä jostakin diskreetistä mahdollisten valintojen joukosta, ja se riippuu signaalijoukosta *v*. Riippuvuus voidaan kuvata funktiolla $d = f(v)$, jota kutsutaan *päätösfunktioksi* (decision function). Hahmontunnistusongelman ratkaiseminen edellyttää sellaisen päätösfunktion konstruoimista, joka ottaa huomioon erottelun mahdollistavat käytännön seikat. Ei ole esimerkiksi mitenkään yksinkertaista valita päätösfunktiota, joka ottaa huomioon tuntemattomat signaalit. Tällöin signaalijoukosta *v* täytyy olla jonkinlainen malli.

Monissa hahmontunnistuksen ongelmissa käsiteltävästä datasta on vain vähän tietoa (esimerkiksi tilastollisia malleja) saatavilla etukäteen. Sen vuoksi päätöksen tekijän täytyy tehdä mahdollisimman vähän ennakko-oletuksia datasta. [Jain *et al.*, 1999]

Jain *et al.* [1999] viittaavat Watanaben ruman ankanpoikasen teoreemaan: ”Sikäli kun käytämme äärellistä määrää predikaatteja erottamaan mitkä tahansa kaksi objektia toisistaan, minkä tahansa kahden objektin jakamien predikaattien määrä on vakio – riippumaton objektien valinnoista.” Jainin *et al.* mukaan tästä seuraa, että kahdesta mielivaltaisesta hahmosta saadaan samankaltaisia, kun niihin koodataan tarpeeksi piirteitä. Sen seurauksena mitkä tahansa kaksi mielivaltaista hahmoa ovat yhtä samankaltaisia, mikäli ei lisäksi käytetä hyväksi jotakin sovellusalueen tietämystä. Esimerkiksi jalkapallossa voidaan olettaa, että maalivahti ei poistu maalialueelta. Tätä heuristista tietoa voidaan hyödyntää maalivahdin ja muiden pelaajien erottamiseen toisistaan. Sovellusalueen tietämystä käytetään implisiittisesti jo valittaessa kyseessä olevaan ongelmaan soveltuvaa klusterointialgoritmia.

Kovalevsky kiinnittääkin huomiota siihen, että hahmontunnistus ei ole erityisen formaali alue. Menetelmät perustuvat yleensä edellä mainitun kaltaisiin heuristiikkoihin ja tilannekohtaisiin ratkaisuihin. Koska useat hahmontunnistusongelmat ovat monimutkaisia, Kovalevsky peräänkuuluttaakin formaalimpaa lähestymistä aiheeseen. Tilanne on nykyään melko samanlainen kuin 1970-luvulla. Vaikka hahmontunnistustekniikoita voidaankin kategorisoida, ovat hahmontunnistussysteemit

useasti vahvasti heuristiikkoihin perustuvia. Varsinkin reaaliaikaisissa järjestelmissä tehokkuusvaatimukset ovat pakottaneet tilannekohtaisiin optimointeihin ja heuristiikkoihin. Esimerkiksi Siermala *et al.* [2000] ovat voineet vähentää käsiteltävien kuvien määrää sen perusteella, että jalkapalloilijat tai pallo eivät voi liikkua tiettyä vauhtia nopeammin. Choi *et al.* [1997] eivät seuraa jalkapalloa kuten pelaajia, vaan liittävät pelaajiin ominaisuuden ”has ball”. Tämä siksi, että pallon seuraaminen on sen pienen koon ja jatkuvan pelaajien taakse peittymisen vuoksi erittäin vaikeata.

2.1. Hahmontunnistustekniikoita

Seuraavaksi esittelen hahmontunnistuksessa yleensä käytettyjä tekniikoita. Lista ei ole täydellinen, mutta antaa hyvän kuvan siitä, millaisia erilaisia lähestymistapoja hahmontunnistuksen ongelmaan on olemassa. Lisäksi kerron jokaisen tekniikan kuvauksen yhteydessä alan aiemmasta tutkimuksesta ja siitä, kuinka olen sitä soveltanut omassa tutkimuksessani.

Tekniikat on mielekästä jakaa kahteen ryhmään: *virtauspohjasiin* (flow based) ja *piirrepohjaisiin* (feature based). Virtauspohjaiset tekniikat perustuvat pikseleiden liikevirtausten seuraamiseen. Kuvan pisteille voidaan laskea liikevektoreita ja näin selvittää kuvassa liikkuvia kohteita. Virtauspohjaisten tekniikoiden ongelmana onkin se, että niillä on vaikea havaita paikallaan olevia objekteja. Siksi ne eivät sovi suoraan tarkasteltavana olevaan ongelmaan. Piirrepohjaiset tekniikat perustuvat kuvan piirteiden havainnointiin. Piirteet voivat olla yksinkertaisesti värejä tai esimerkiksi kuvaan sovitettavia rautalankamalleja kasvonpiirteistä.

Tekniikoiden hienompaa jaottelua vaikeuttaa se, että alan kirjallisuudessa samoista tekniikoista puhutaan useasti eri nimillä ja luonnollisesti tästä aiheutuu myös monesti päällekkäisyyksiä. Esimerkiksi Watanabe [1970] kirjoittaa siitä, kuinka geneerisiä termit ”piirre” ja ”hahmo” ovat, ja kuinka se vaikeuttaa niiden määrittelyä. Siksi olen jaotellut tekniikat tässä tutkielmassa melko karkealla tavalla. Aluksi esittelen piirrepohjaiset tekniikat - *mallin sovittaminen* (template matching), *piirteiden erottelu* (feature extraction) ja *erotuskuvat* (frame differencing). Sen jälkeen esittelen virtauspohjaiset tekniikat - *optinen virtaus* (optical flow) ja *aktiiviset ääriviivat* (active contours).

2.1.1. Mallin sovittaminen

Mallin sovittaminen perustuu siihen, että on olemassa jokin malli etsittävistä hahmoista, jota sitten yritetään sovittaa kuvaan. Mallina on yleensä jokin tapa luokitella kuva väriarvojen tai kirkkauden perusteella. *Yksinkertaisessa rajaamisessa* (simple thresholding) ja *automaattisessa rajaamisessa* (automatic thresholding) objektit identifioidaan vertaamalla kuvan pisteiden väriarvoja määrättyyn raja-arvoon. Raja-arvo

voi olla yksittäinen arvo tai värikomponenttien hajonta. Raja-arvona voi toimia myös joukko kuvia (malleja) etsittävästä objektista, joiden arvoja vertaillaan kuvamateriaaliin. Esimerkiksi Choi *et al.* [1997] tunnistavat jalkapalloilijoita käyttämällä malleja tyypillisistä pelaajan kuvista. *Pikselien samankaltaisuuteen* (pixel similarity) perustuvissa systeemeissä kuvasta valitaan piste ja laajennetaan aluetta etsimällä pisteen ympäriltä samankaltaisia pisteitä. Tätä tekniikkaa kutsutaan myös *alueen kasvattamiseksi* (region growing).

Yksinkertainen rajaaminen

Yksinkertaisessa rajaamisessa pikseli luokitellaan kuuluvaksi johonkin objektiin, mikäli jokin seuratuista väriarvoista on yli / alle tietyn rajan. Ongelmana on se, että useasti osa taustaa ja muita objekteja tunnistetaan myös samaan objektiin kuuluvaksi. Lisäksi kuvia otetaan useasti eri olosuhteissa, joten kirkkausarvot eivät ole aina samoja. Myös varjot ja heijastukset sotkevat tunnistusta. Valaistuksen ongelmaa ovat käsitelleet mm. Kehtarnavaz ja Rajkotwala [1997], Agbinya ja Rees [1999], Chang ja Lee [1997] sekä Tan *et al.* [1999]. Yleisesti käytetty ratkaisu valaistuksen ongelmaan on suodattimien käyttö. Suodattimista kerrotaan myöhemmin kappaleessa 2.2.

Yksinkertainen rajaaminen voidaan suorittaa myös hyödyntämällä *värihistogrammien käänteisprojektiota* (color histogram backprojection). Tällöin ei vertailla yksittäistä väriarvoa vaan koko objektin sisältämien värien hajontaa. Etsittävistä objekteista pitää siis olla malli, joten systeemille täytyy ”näyttää” seurattava objekti. Tätä tekniikkaa on soveltanut esimerkiksi Agbinya ja Rees [1999]. He esittävät Swainin ja Ballardin kuvauksen värihistogrammien käänteisprojektiosta. Olkoon M kiinnostuksen kohteena olevan objektin histogrammi ja olkoon I kuvan histogrammi. Lasketaan suhde $min(M_i / I_i, 1)$, missä i tarkoittaa *histogrammipylvään* (histogram bin) indeksiä. Saatu suhde on luotettavuusarvo, kuinka karakteristinen väri i on mallille. Tämän jälkeen kuvalle konstruoidaan harmaasävyinen käänteisprojektiokuva korvaamalla kaikki i:n väriset pikselit sen skaalatulla luotettavuusarvolla $255 * min(M_i / I_i, 1)$. Käänteisprojektiokuvassa malli näkyy kirkkaana “möykkynä”. Värihistogrammien käänteisprojektion hyödyntäminen on ainakin VideoAnalyzerin tapauksessa ongelmallista, sillä tarkoituksena on nimenomaan tunnistaa pelaajat ilman käyttäjän avustusta.

Yksinkertaista rajaamista ovat hyödyntäneet mm. Marchant *et al.* [1998] kasvien ja ruohojen erotteluun ja Tan *et al.* [1999] ihmisen asennon tarkkailuun. Molemmissa tutkimuksissa seurattavat hahmot ovat olleet huomattavasti suurempia kuin VideoAnalyzerin tapauksessa. Yksi VideoAnalyzerin leimaa-antava piirre onkin se, että sen etsimät hahmot ovat niin pieniä, että useimmissa muissa tutkimuksissa niitä pidettäisiin roskina tai meluna ja pyrittäisiin suotimaan pois.

Automaattinen rajaaminen

Automaattinen rajaaminen on yksinkertaisen rajaamisen kehittyneempi versio. Siinä raja-arvoja päivitetään automaattisesti. Tämä tapahtuu tutkimalla kuvan histogrammia. Nyt histogrammia ei käytetä objektien, vaan taustan etsimiseen. Choi *et al.* [1997] ovat käyttäneet histogrammia taustan poistamiseen jalkapallovideosta. Näin jäljelle jäävät vain pelaajat. Tätä olisi mahdollista käyttää myös VideoAnalyzerissä. Olen kuitenkin pyrkinyt pitämään VideoAnalyzerin niin nopeana kuin mahdollista. Tästä syystä taustan automaattista päivittämistä ei ole implementoitu.

On huomattava, että sekä yksinkertaisessa rajaamisessa että automaattisessa rajaamisessa tulokseksi saadaan kelpuutettuja pikseleitä. Tarvitaan siis vielä jokin eri tekniikka varsinaiseen klusterointiin. Kelpuutettuja pikseleitä voidaan yhdistellä esimerkiksi, mikäli niiden välinen etäisyys on pienempi kuin asetettu raja-arvo. Myös polkujen etsintään käytettävät etäisyysmitat ovat sovellettavissa klusterointiin. Niistä kerron polkujen etsinnän yhteydessä luvussa 2.3.

Pikselien samankaltaisuus* eli *alueen kasvattaminen

Pikselien samankaltaisuuteen perustuva luokittelu tapahtuu kasvattamalla jonkin kelpuutetun pikselin aluetta. Tarkasteltavan pikselin kelvollisuus voidaan tarkistaa esimerkiksi yksinkertaisella tai automaattisella rajaamisella. Luokittelu tapahtuu klusteroimalla *n* samankaltaisinta ympäröivää pikseliä yhdeksi klusteriksi. Toinen tapa on käydä kelpuutetun pikselin ympärys läpi – esimerkiksi 4 tai 8 lähintä ympäröivää pikseliä – ja mikäli samankaltaisia pikseleitä löytyy, etsintää jatketaan rekursiivisesti niihin. Jälkimmäisen tavan etuna on se, että etsittävät hahmot voivat olla mielivaltaisen kokoisia ja muotoisia.

Tan *et al.* [1999] ovat havainneet, että segmentoitaessa suuria pintoja ei-rekursiivisella alueen kasvattamiseen perustuvalla menetelmällä, ne jakaantuvat helposti pienempiin marmorimaisiin alueisiin. Ongelmaa voidaan helpottaa lieventämällä samankaltaisuusvaatimusta, mutta silloin myös objektiin kuulumattomia osia tulee mukaan. Tämä ei ole ongelma VideoAnalyzerin tapauksessa, sillä objektit ovat niin pieniä, että ne eivät yleensä jakaannu useisiin osiin.

Pikselien samankaltaisuuteen perustuva segmentointi on kätevää, sillä tuloksena muodostuu valmiita klustereita. Alueen kasvattaminen onkin erittäin tehokas tekniikka, sillä klusterointi tapahtuu siinä pikselien kelpuuttamisen yhteydessä ilman laskennallista lisätaakkaa. VideoAnalyzerin nykyinen versio perustuu yksinkertaiseen rajaamiseen, rekursioon perustuvaan alueen kasvattamiseen ja reunojen tunnistamiseen.

2.1.2. Piirteiden erottelu

Piirteiden erottelu perustuu siihen, että kuvista etsitään tunnistettavia piirteitä ja seurataan niitä kuvasta toiseen. Yleensä käytettyjä tunnistettavia piirteitä ovat esimerkiksi nurkat, suorat linjat, kaaret tai vapaamuotoiset käyrät. Myös monimutkaisempia piirteitä on hyödynnetty. Esimerkiksi Li ja Wang [1999] ovat käyttäneet piirteiden erotteluun kolmiulotteisia malleja. Tärkeintä piirteiden erottelussa on *poissulkemisen periaate* (principle of exclusion). Kun kahta ruutua verrataan toisiinsa, sama piirre saa esiintyä kussakin ruudussa vain kerran [Pla and Marchant, 1997].

Rosin [1997] mainitsee, että yleensä *reunojen tunnistajat* (edge detectors) löytävät objektien oikeiden reunojen lisäksi muitakin reunoja. Tämä johtuu kuvan häiriöistä ja kirkkauden vaihteluista. Siyal ja Fathy [1999] ovat kehittäneet systeemiä liikenteen seurantaan. He viittaavat aiempaan tutkimukseensa kirjoittaessaan, että reunojen tunnistaminen on vähiten herkkä tekniikka valaistuksen vaihteluille, sillä autojen reunat kärsivät vähemmän valaistuksen vaihteluista kuin muut auton osat. Tämä on varmasti yleistettävissä muihinkin tilanteisiin. Vaikka virhehavaintoja olisikin muita tekniikoita vähemmän, virhehavainnot kuitenkin sotivat poissulkemisen periaatetta vastaan. Sen lisäksi, että virheelliset reunat häiritsevät tunnistusta, ne lisäävät laskennallista taakkaa. Pla ja Marchant toteavatkin, että yleensä piirteiden etsinnän jälkeen suoritetaan jokin funktio, jolla poistetaan virheellisiä löydöksiä ja yhdistellään lähekkäin olevia piirteitä.

Piirteiden erottelun hyödyntäminen VideoAnalyzerissä on mahdotonta, sillä etsittävät objektit ovat niin pieniä, että niistä on mahdotonta löytää mitään edellämainittuja piirteitä. Muita piirteitä, kuten objektin kokoa, voidaan kyllä käyttää hyväksi polkujen etsinnässä päällekkäisyyksien tunnistamiseen.

2.1.3. Erotuskuvat

Erotuskuvat perustuvat siihen, että on olemassa *vertailukuva* (reference image), johon jotakin toista kuvaa verrataan. Voidaan esimerkiksi ottaa kuva tyhjästä jalkapallokentästä ja vähentää se kuvasta, jossa on mukana myös pelaajia. Näin taustan pitäisi hävitä ja eroavaisuuksien (eli pelaajien) pitäisi näkyä suurina amplitudieroina. Siyal ja Fathy [1999] puolestaan selvittävät ensimmäisen vertailukuvan laskemalla keskiarvon sadasta ruudusta.

Chang ja Lee [1997] määrittelevät erotuskuvan $d_{k,\,k+1}$ kuvien *k* ja *k+1* väillä seuraavasti:

$$d_{k,k+1}(i,j) = \begin{cases} 1 & jos \parallel f(i,j,k) - f(i,j,k+1) \parallel > h, 0 \le i < x, 0 \le j < y \\ 0 & muuten \end{cases}$$

missä η on eroavuuden raja-arvo, *x* ja ja *y* ovat kuvan resoluutiot X- ja Y-akseleilla ja *f(i, j, k)* on kuvan kirkkaus koordinaateissa *(i, j)* kuvassa *k*. Pikselien välinen ero || *f(i, j, k) - f(i, j, k+1)* || on etäisyysmitta – esimerkiksi euklidinen etäisyys RGB-väriavaruudessa. Kerron lisää etäisyysmitoista polkujen etsinnän yhteydessä alaluvussa 2.3.

Edellisten tekniikoiden tapaan myös erotuskuvat ovat herkkiä valaistuksen vaihteluille. Kehtarnavaz ja Rajkotwala [1997] ovat kehittäneet järjestelmän jalankulkijoiden tarkkailuun. He ovat pyrkineet ratkaisemaan valaistusongelman käsittelemällä kuvan palasissa ja päivittämällä vertailukuvaa automaattisesti. Kuva jaetaan esimerkiksi 16 x 8 pikselin palasiin, joista lasketaan keskimääräinen kirkkaus. Näin saadaan minimoitua häiriöiden vaikutus. Lisäksi vertailukuva lasketaan muutaman minuutin keskiarvojen perusteella. Näin liikkuvien objektien ja valaistusolosuhteiden (esimerkiksi liikennevalojen) vaikutukset saadaan karsittua pois. Vertailukuvan *B* laskennassa Kehtarnawaz ja Rajkotwala ovat päätyneet seuraavaan funktioon:

$$B(t; i, j) = (1-a) \cdot B(t-1; i, j) + a \cdot I(t; i, j),$$

missä *t* tarkoittaa aikaa tai ruudun numeroa, *(i, j)* pikselin x- ja y-koordinaatteja ja *I* itse kuvaa. Luku *a* esittää yhden ruudun vaikutusta vertailukuvan muodostamiseen. Jos vertailukuva lasketaan esimerkiksi 5 minuutin ajalta (300 s) nopeudella 3 ruutua / s, *a* on $1 / (3 \times 300) \approx 0.001$.

Myös Siyal ja Fathy [1999] ovat käyttäneet vastaavaa automaattista vertailukuvan päivitysfunktiota liikenteen tarkkailuun. He ovat kokeilleet kolmea eri tekniikkaa vertailukuvan päivitykseen: pikselitason päivitystä, Kehtarnavazin ja Rajkotwalan systeemiä vastaavaa palasissa päivitystä ja koko kuvan päivitystä. Pikselitasolla tietty pikseli lasketaan vertailukuvaan, mikäli se on muuttunut vähemmän kuin 10 % edellisestä kuvasta. Palasiin perustuvassa tekniikassa päivitys tapahtuu, kun pala ei sisällä liikettä tai pysäköityä objektia (autoa). Kuvatason tekniikassa päivitys tapahtuu, kun kuva ei sisällä raskasta liikennettä, eli jos alle puolet palasista ei sisällä liikettä tai paikallaan olevia objekteja. Tuloksista käy ilmi, että tunnistustarkkuus on pikselitason päivityksessä jopa kolme kertaa tarkempi kuin kuvatason päivityksessä. Luonnollisesti tarkemmat tulokset tarkoittavat myös huomattavasti raskaampaa laskennallista taakkaa.

VideoAnalyzerissä palasiin jakoa ei voida suorittaa, koska pelaajat karsiutuisivat pois pienen kokonsa vuoksi häiriöinä. Erotuskuvien hyödyntäminen sinänsä olisi täysin mahdollista ja olisi tekniikan nopeuden puolesta jopa suositeltavaa. Erotuskuvien implementointi onkin yksi tulevaisuuden jatkokehityksen aihe.

Vertailukuvana on mahdollista käyttää myös edellistä kuvaa. Kun tausta ja kamera ovat staattisia, vain liikkeet erottuvat. Tällaista erotuskuvien käyttöä voisi pitää eräänlaisena

optisen virtauksen ”köyhän miehen versiona”. Etuna on luonnollisesti se, että tällöin valaistusolosuhteiden muutokset eivät juuri vaikuta tulokseen, koska vertailukuva on vain yhden ruudun jäljessä nykyisestä. Haittapuolena on se, että tällainen vertailukuvan päivitys kärsii optisen virtauksen ongelmista. Jotkut VideoAnalyzerin mallit hyödyntävät differenssimatriisia, joka perustuu peräkkäisten kuvien erotuskuviin (ks. luku 3.1.2).

2.1.4. Optinen virtaus

Optinen virtaus ja aktiiviset ääriviivat ovat ns. *gradienttimenetelmiä* (gradient method). Kim *et al.* [1997] viittaavat artikkelissaan Hornin ja Schunkin luonnehdintaan gradienttimenetelmistä. Hornin ja Schunkin mukaan gradienttimenetelmät perustuvat *avaruus-ajalliseen rajoiteyhtälöön* (spatio-temporal constraint equation), joka voidaan käsittää kahden tuntemattoman muuttujan *u* ja *v* (vaakasuuntainen ja pystysuuntainen nopeus) lineaariseksi yhtälöksi. Camuksen [1997] mukaan gradienttimenetelmät ovat kuuluisia häiriöherkkyydestään.

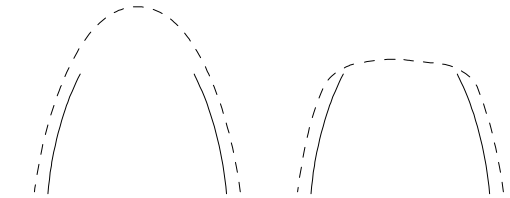
Optinen virtaus syntyy liikkeestä. Kuvasta ei siis tunnisteta hahmoja vaan liikettä. Optinen virtaus perustuu siihen, että peräkkäisistä kuvista etsitään pikseleiden liikevektoreita, eli mihin suuntaan kukin pikseli / pikselirypäs on liikkeessä. Tätä tekniikkaa käytetään yleensä, kun kuvan tausta liikkuu. Siksi optinen virtaus ei sovellu tämän tutkimuksen aihepiiriin.

2.1.5. Aktiiviset ääriviivat

Aktiiviset ääriviivat tunnetaan paremmin nimellä *madot* (snakes). Madot on suosittu tekniikka hahmojen reunojen tunnistamiseen. Reunojen tunnistaminen tapahtuu menetelmällä, joka on monesti tehokkaampi kuin perinteiset reunojentunnistustekniikat [Gunn, 1996]. Matojen sovellusalueita ovat olleet lääketieteellinen kuva-analyysi, liikkeen tunnistus sekä ihmisten kasvojen tunnistaminen [Tohka, 2001; Tohka 1998].

Tohka kertoo madon intuitiivisen analogian tulevan fysiikasta. Siinä elastinen ja kiinteä narun tapainen kappale asetetaan potentiaali-kenttään, ja sen annetaan asettua luonnolliseen paikkaansa tässä kentässä. Hän havainnollistaa periaatetta kuminauhalla, joka venyy sormien muodostamassa energia-kentässä. [Tohka, 1998]

Yksinkertaistettuna mato voidaan käsittää hahmon ääriviivojen läheisyydessä sijaitsevien pisteiden ketjuksi $V = \{v_1, v_2, v_3, \ldots, v_n\}$. Pisteketjulla on kokonaisenergia $E(V)$, joka muodostuu ulkoisesta energiasta E_{ext} ja sisäisestä energiasta E_{int}. Pitääkseen energiansa minimissään madon on pysyttävä sille määräytyneessä muodossa ja oikeassa suhteessa kuvassa olevaan informaatioon.



Kuva 2.3. Sisäisen energian osatavoitteet. [Gunn, 1996]

Sisäinen energia voidaan jakaa kahteen osatavoitteeseen, joista toinen tavoite pyrkii pitämään madon ääriviivat elastisina ja toinen pyrkii vastustamaan taipumista. Kuva 2.3 esittää osatavoitteiden vaikutusta. Kuvan a-kohdan osatavoite pyrkii välttämään terävien kulmien muodostumista. Näin ollen mato seuraa ”katkenneen paraabelin” alkuperäistä muotoa. Kuvan b-kohdassa osatavoite pyrkii elastisuuteen; aivan kuin kuvion ympärille olisi laitettu kuminauha. Ulkoinen energia määrittelee piirteet, jotka vetävät madon ääriviivaa puoleensa.

Liikkuvan kuvahahmon seurannassa pitää antaa pistejoukolle alkuarvot jonnekin hahmon äärirajojen läheisyyteen, minkä jälkeen lähdetään minimoimaan madon energiafunktiota laskevan gradientin menetelmällä. Kun minimi on saavutettu, on pistejoukolla uudet positiot, jotka ovat luultavimmin hahmon äärirajoilla. Tämän jälkeen voidaan ottaa uusi kuva, jossa hahmo on ehkä liikkunut äskeisestä positiosta jonkin verran. Vanhat arvot toimivat nyt alkuarvoina uudelle minimoinnille ja toiminta jatkuu minimoinnilla. [Gunn, 1996]

2.2. Suodattaminen

Suodattamisen tarkoitus on yleensä korostaa kuvan joitakin piirteitä. Suodattaminen ei itsessään varsinaisesti kelpaa segmentointiin. Siitä voi kuitenkin olla apua kuvan esiprosessoinnissa ennen segmentointia. Hawkins [1970] mainitsee esimerkkinä, että haluttu tieto voi olla tietyllä taajuusalueella, jonka muut taajuudet piilottavat, tai tiedon amplitudi voi olla niin pieni, että sitä on syytä skaalata suuremmaksi. Tan *et al.* [1999] kertovat, että rajaaminen on useasti ongelmallista, koska eri objektien pikselien arvot voivat olla melko samanlaisia. Chang ja Lee [1997] taas pyrkivät parantamaan segmentointia suodattamalla pois jo kelpuutettuja pikseleitä, mikäli niiden ympärillä ei ole tietyn raja-arvon ylittävää määrää muita kelpuutettuja pikseleitä. Kuvan valaistus voi myös olla erilainen kuvan eri osissa, jolloin suodattamisella pyritään ”tasapäistämään” semanttisesti samankaltaisia alueita.

On vaikea tehdä eroa suodattamisen ja yksinkertaisen rajaamisen välille. Voidaanhan ajatella, että yksinkertaisessa rajaamisessa korostetaan kuvan piirteitä luokittelemalla pikseleitä väriarvojen perusteella. Erona voidaan kuitenkin pitää sitä, että

yksinkertaisessa rajaamisessa kuvaa ei muokata. Suodattamisessa taas kuvaa muokataan konkreettisesti.

Spatiaalisen taajuussuodattamisen (spatial frequency filtering) avulla kuvasta voidaan saada irti sellaista informaatiota, jota ihmissilmällä on mahdotonta havaita. Suodattamisen avulla voidaan rekonstruoida valokuva, joka on mennyt pilalle, kun kamera on heilahtanut kuvaushetkellä. Toiseksi suodattamisella voidaan selkeyttää matalakontrastisia röntgenkuvia, ja kolmanneksi sen avulla voidaan ”nähdä esteen läpi” (este voi olla esimerkiksi savua) ja niin edelleen.

Jos $f(x, y)$ esittää mitä tahansa kaksiulotteista dataa, $F(p, q)$ on sen Fourier-spektri, joka on suorakulmaisissa koordinaateissa määriteltynä:

$$F(p,q) = \int\int f(x,y)e^{-j(px+qy)}dx\,dy,$$

missä $p, q = 2p / l$ ja l on aallonpituus etäisyysyksiköissä. Mikäli etsittävä tieto on kuvan pienissä yksityiskohdissa (pieni l) ja ne ovat suurten ei-kiinnostavien alueiden peittämänä (suuri l), tarvitaan *ylipäästösuodatinta* (high-pass filter). Ylipäästösuodatin korostaa $F(p, q)$:n korkeiden taajuuksien arvoja suhteessa matalien taajuuksien arvoihin. [Hawkins, 1970]

2.3. Polkujen etsintä

Koska virtauspohjaiset tekniikat perustuvat liikkeen seurantaan, ei niiden yhteydessä puhuta erikseen polkujen etsinnästä. Piirrepohjaisten tekniikoiden avulla identifioidut objektit ovat kuitenkin sikäli hyödyttömiä, että niillä ei ole yhteyttä eri kuvien välillä. Tämän vuoksi löydettyjen objektien välille pitää saada yhteys polkujenetsintäalgoritmilla. Algoritmi laskee objektien liikeradat käyttämällä piirteisiin liitettyjä ankkuripisteitä [Pla and Marchant, 1997]. Ankkuripiste voi olla esimerkiksi objektin vasen alanurkka tai objektin keskipiste.

Havaintojen yhdistelyyn käytetään jotakin etäisyysmittaa. Hyvin yleinen tapa on käyttää euklidista etäisyyttä etäisyysmittana. Jain *et al.* [1999] viittaavat artikkeliin, jossa Mao ja Jain kertovat euklidisen etäisyysmitan toimivan hyvin, jos datassa on kompakteja ja eristettyjä klustereita. Jain *et al.* lisäävät, että muussa tapauksessa suurimmat piirteet tapaavat dominoida muita.

Agbinya ja Rees [1999] käyttävät hieman erilaista etäisyysmittaa. Olkoon d_i on etäisyys i:nnen ehdokkaana olevan objektin ja objektin edellisen kuvan havainnon välillä. A_i on ehdokasobjektin alue ja A_p edellisen kuvan havainnon alue. Nyt ehdokasobjekteille voidaan laskea etäisyysmitta:

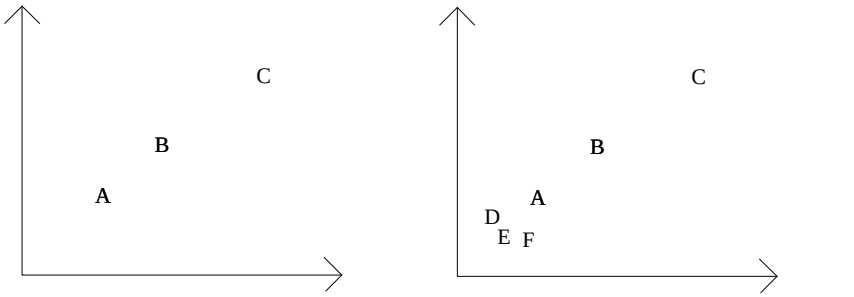
$$S_t = d_i|A_p - A_i|.$$

Agbinyan ja Reesin etäisyysmitta ottaa siis huomioon myös objektien koon. Mitä suurempi kokoero edellisellä havainnolla ja ehdokkaana olevalla objektilla on, sitä ”kauempana totuudesta ollaan”. Yleensä ehdokasobjekti, jolla on pienin S_t, on seurattava objekti. Koska näin ei aina kuitenkaan ole, käyttävät Agbinya ja Rees myös liikkeen ennustamista ehdokasobjektien valinnassa. Ehdokasobjektin pitää liikkua samaan suuntaan kuin edellisessä kuvassa havaitun objektin.

Kaikkein yksinkertaisin päätöksentekokriteeri on verrata objektien välistä etäisyyttä johonkin ennalta määriteltyyn raja-arvoon. Tämä tapa on kuitenkin niin herkkä virheille, että sitä ei kannata ainakaan polkujen etsinnän yhteydessä hyödyntää. Jos vertailu suoritetaan johonkin raja-arvoon, muiden klustereiden vaikutus jää kokonaan huomioimatta. Klusteri voi esimerkiksi olla raja-arvoetäisyyden sisällä, mutta silti kauempana kuin monet muut klusterit. Jain *et al.* [1999] esittelevät Gowdan ja Krishnan kehittämän *yhteisen naapurin etäisyyden* (mutual neighbour distance (MND)), joka ottaa huomioon lähellä olevat klusterit. Se määritellään kaavalla

$$MND(x_i, x_j) = NN(x_i, x_j) + NN(x_j, x_i),$$

missä $NN(x_i, x_j)$ tarkoittaa sitä, kuinka monenneksi lähin naapuri x_i on x_j:lle. Kuvassa 2.4 on esimerkki. Kuvan a-kohdassa *A*:n lähin naapuri on *B* ja *B*:n lähin naapuri on *A*. Siis $NN(A, B) = NN(B, A) = 1$ ja $MND(A, B) = 2$. Pisteen *C* lähin naapuri on *B*, eli $NN(B, C) = 1$, mutta $NN(C, B) = 2$, joten $MND(B, C) = 3$. Kohdassa b kuvaan on lisätty pisteet *D*, *E* ja *F*. Nyt $MND(B, C) = 3$ kuten kohdassa a, mutta $MND(A, B) = 5$. *MND* pisteiden *A* ja *B* välillä on kasvanut, vaikka ne eivät ole liikkuneet; kasvu johtuu *A*:n läheisyyteen lisätyistä pisteistä *D*, *E*, *F*.



(a) Pisteet A ja B ovat samankaltaisempia kuin A ja C.

(b) Pisteiden lisäyksen jälkeen B ja C ovat enemmän samankaltaisia kuin B ja A.

Kuva 2.4. Yhteisen naapurin etäisyys. [Jain *et al.*, 1999]

Kun klustereiden välinen etäisyysmitta on valittu, tarvitaan tekniikkaa, jolla polku muodostetaan. Yleinen tapa on käyttää *lähimmän naapurin* (nearest neighbour) menetelmää. Siinä etsitään klusteria aidosti lähimpänä oleva klusteri, eli sellainen klusteri, joka on lähempänä kyseessä olevaa klusteria kuin mitään muuta. Tämän

jälkeen löydetty klusteri valitaan uudeksi vertailukohteeksi ja edellinen klusteri poistetaan läpikäytävien listasta. Prosessia jatketaan, kunnes sopivia klustereita ei enää löydy. VideoAnalyzerissä käytetään juuri lähimmän naapurin tekniikkaa erilaisilla heuristiikoilla terästettynä.

Vaikka VideoAnalyzerissä ei hyödynnetä liikkeen ennustamista, sitä on mahdotonta ohittaa, sillä se esiintyy muodossa tai toisessa lähes kaikissa tutkimuksissa. Yleisin käytetty tekniikka liikkeiden ennustamiseen lienee Kalman-suodattaminen. Muun muassa Yeasin ja Chaudhuri [2000], Choi *et al.* [1997], Li ja Wang [1999] ja Marchant *et al.* [1998] ovat hyödyntäneet sitä tutkimuksissaan. Kalman-suodattamisessa lähdetään siitä oletuksesta, että data on häiriöistä. Siksi se soveltuu hyvin liikkeen ennustamiseen videokuvasta. Kalman-suodattimien ongelmana on niiden laskennallinen raskaus. Ne koostuvat joukosta lineaarisia painottamattomia tilanennustusalgoritmeja.

Yksinkertaisempi tapa ennusta liikettä on laskea objektin kahden edellisen kuvan perusteella saatu nopeus ja olettaa liikkeen jatkuvan samana seuraavassakin kuvassa. Agbinya ja Rees käyttävät liikkeen ennustamiseen seuraavaa kaavaa:

$$v_x = xpos(j-1) - xpos(j-2)$$

$$v_y = ypos(j-1) - ypos(j-2),$$

missä v_x ja v_y ovat objektin nopeuden x- ja y-komponentit, *j* on ajan hetki ja *xpos*, *ypos* ovat objektin edellisten havaintojen keskipisteet. Myös Pla ja Marchant [1997] ovat käyttäneet vastaavaa menetelmää.

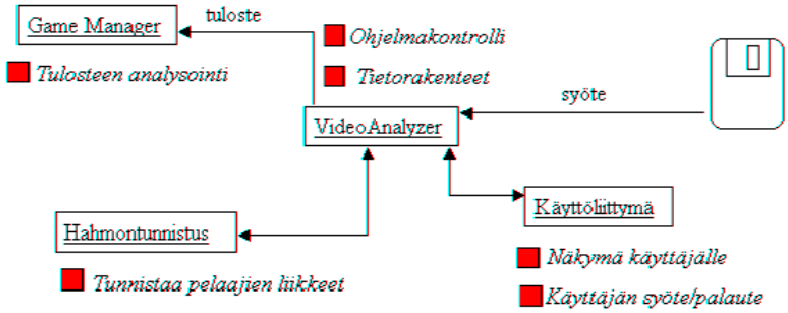
Suurin syy virheellisten polkujen löytymiseen on objektien päällekkäin meneminen. Jos klusterointialgoritmi tulkitsee päällekkäiset tai erittäin lähekkäin olevat objektit yhdeksi, polkujenetsintäalgoritmin näkökulmasta toinen objekteista häviää (ainakin hetkellisesti) kuvasta. Agbinya ja Rees lähestyvät ongelmaa siten, että mikäli objekti häviää, edellisen havainnon mallia säilytetään ja käytetään kunnes objekti ilmestyy taas näkyviin. Heidän mukaansa ratkaisu toimii, ellei objekti ole pitkään kadoksissa.

Kun sovellusalueena on jalkapallo, pelaajien vaatteet ovat ainakin joukkueittain samanväriset. Vaikka hahmontunnistussysteemi käyttäisikin objektien väri-informaatiota hyväkseen, päällekkäisyystilanteen päätyttyä objektien oikea identifiointi on erittäin hankalaa. Pelaajia on vaikea erottaa toisistaan. Jalkapallon tapauksessa useampienkin pelaajien päällekkäisyys on tavallista. Varsinkin VideoAnalyzerin kameran paikalle asettamat vaatimukset ovat omiaan aiheuttamaan paljon päällekkäisyyksiä. Choi *et al.* [1997] ovatkin tyytyneet käsittelemään vain vastakkaisten joukkueiden pelaajien välisiä päällekkäisyyksiä joukkueiden pelipaitojen väri-informaatiota hyödyntämällä.

3. VideoAnalyzer

VideoAnalyzer on toiminut testialustana erilaisille tekniikoille. Se on pyritty rakentamaan mahdollisimman modulaarisesti, jotta palasia voitaisiin vaihtaa ”plug and play”-menetelmällä. Kuvassa 3.1 on korkean tason kuvaus ohjelman rakenteesta. VideoAnalyzer on ohjelman pääkomponentti, joka kontrolloi muita komponentteja. Käyttöliittymän avulla käyttäjä syöttää analyysiin tarvittavia tietoja ja muokkaa löydettyjä polkuja. Jalkapallovideo tulee ohjelmalle syötteenä AVI-tiedostosta. Tulevaisuudessa video voi mahdollisesti tulla suoraan videoilta digitointikortin avulla. Hahmontunnistuskomponentti hoitaa klusteroinnin ja polkujen etsimisen. Löydetyt polut tulostetaan tiedostoon KIHU:n GameManageria varten.

Vaikka koko sovellus on ollut kehityksen alla, pääpaino on ollut nimenomaan erilaisten algoritmien ja tietorakenteiden kehittämisessä. Algoritmien evaluointia varten olen lisännyt VideoAnalyzeriin ominaisuuksia, jotka lopulta mahdollistaisivat myös tulosteen analysoinnin ilman GameManageria. Esimerkiksi löydetyt polut esitetään ”tikku-ukkograafina” (kuva 3.7). Tämän yhteyteen olisi mahdollista lisätä myös pelin taktiseen analysointiin liittyviä ominaisuuksia.



Kuva 3.1. Sovelluksen arkkitehtuuri on suunniteltu soveltaen MVC (Model – View – Controller) – mallia.

Tässä luvussa käsitellään VideoAnalyzerissä käytettyjä tekniikoita ja tutkimuksessa kohdattuja ongelmia. Monet käytetyistä tekniikoista perustuvat alun perin Siermalan [1999] ja Siermala *et al.*:n [2000] tutkimukseen. Alaluvussa 3.1 käydään läpi analyysin eri vaiheet, jotka ovat esiprosessointi, klusterointi ja polkujen etsintä. Alaluvussa 3.2 esitellään VideoAnalyzerissä käytettyjä tietorakenteita ja niihin liittyviä ongelmia. Alaluvussa 3.3 käsitellään objektien paikantamisen kysymystä: miten videokuvasta voidaan selvittää pelaajien todelliset sijainnit.

3.1. Analyysin eri vaiheet

Videon analysointi koostuu kolmesta vaiheesta: esiprosessoinnista, klusteroinnista ja polkujen etsinnästä. Esiprosessoinnissa kuvaa on tarkoitus kohentaa siten, että etsittävät hahmot erottuvat siitä paremmin. Tämä voi tapahtua suodattamalla häiriöitä tai esimerkiksi korostamalla hahmojen piirteitä. Klusteroinnissa video käydään ruutu

ruudulta läpi ja etsitään hahmoja. Polkujen etsimiseksi klusteroinnissa löydetyt yksittäisten ruutujen havainnot yhdistellään poluiksi.

Esiprosessointi voi olla osana klusterointia, mutta klusteroinnin ja polkujen etsinnän on tarkoitus olla täysin toisistaan erillään. Tämä sen vuoksi, että klusterointi suoritetaan täysin ilman käyttäjän avustusta. Polkujen etsintä pyritään suorittamaan myös automaattisesti, mutta koska kaikkia pelaajia ja polkuja ei kuitenkaan yleensä löydetä, tarvitaan myös käyttäjän apua. On mielekästä, että käyttäjän apua pyydetään vasta sitten, kun kaikki automaattisesti prosessoitava data on käsitelty. VideoAnalyzerin ensimmäisissä toteutuksissa kaikki vaiheet tehtiin yhtä aikaa. Siksi ne eivät soveltuneet reaaliaikaiseen analyysiin. Myöhemmissä toteutuksissa reaaliaikaisuuden mahdollisuus on pyritty ottamaan huomioon.

Tulevien kappaleiden rakenne mukailee nykyistä toteutusta. Analyysin jako eri vaiheisiin on mahdotonta VideoAnalyzerin ensimmäisten toteutusten osalta, mutta olen pyrkinyt käymään ne läpi mahdollisimman sujuvasti. Vaiheisiinjakoa tukee se, että eri vaiheissa käytetään erilaisia tekniikoita. Eriteltyinä niitä on helpompi seurata rinnakkain luvun 2 kanssa.

Analyysi lähtee käyntiin sillä, että käyttäjä syöttää ohjelmalle tietoja kentästä ja kameran sijainnista. Kamerasta kerrotaan, kuinka korkealla se on ja mikä sen etäisyys kentän alkupäätyyn on. Lisäksi kerrotaan arvio siitä, kuinka pitkä matka kamerasta on kuvan alkuun (vasempaan alalaitaan). Kentästä kerrotaan sen leveys ja pituus sekä määritellään seuraavat kentän kiintopisteet videosekvenssin ensimmäisestä ruudusta: vasen alalaita, keskiviiva, vasen ylälaita, oikea ylälaita ja kohta, missä kenttä katkeaa oikeassa reunassa. Kiintopisteet on merkitty punavalkoisilla lipuilla kuvassa 3.2. Näiden tietojen syöttämisen jälkeen analyysi etenee seuraavissa kappaleissa esitetyllä tavalla.

Vaikka tässä luvussa esitellään kolme analyysin vaihetta, vaiheita on itse asiassa neljä. Esiteltävät kolme vaihetta muodostavat automaattisen videoanalyysin osuuden, mutta automaattisen analyysin jälkeen tarvitaan myös käyttäjän apua virheellisten havaintojen ja polkujen muokkaamisessa sekä pelaajien nimeämisessä. Tätä vaihetta kutsutaan jälkiprosessoinniksi. Jälkiprosessointivaihe muistuttaa vektoripiirto-ohjelman käyttöä. Tässä tapauksessa tosin X- ja Y-ulottuvuuksien lisäksi ohjelmassa on kolmantena ulottuvuutena aika. Analyysivaiheessa löydettyjä polkuja voidaan yhdistää, katkaista, poistaa, liikuttaa ja nimetä tietylle pelaajalle kuuluvaksi. Myös täysin uusia polkuja voidaan luoda. Joustava polkujen muokkausmahdollisuus on välttämätön ominaisuus, sillä automaattinen videoanalyysi ei ole lähes koskaan täydellinen. Jälkiprosessointi tapahtuu edellä mainitussa ”tikku-ukkograafissa” (kuva 3.7).

Kuva 3.2. VideoAnalyzerin parametrien asetus.

3.1.1. Esiprosessointi

Esiprosessoinnissa kuvaa käsitellään, jotta siitä erotettaisiin paremmin yksittäiset hahmot. Koska VideoAnalyzerin tapauksessa etsittävät hahmot ovat niin pieniä, että niitä on lähes mahdotonta erottaa kuvan häiriöistä, ei suodattimien käytöstä ole juuri apua. Klusterointialgoritmin nykyinen versio perustuu tosin osittain reunojen tunnistamiseen. Kuten alaluvussa 2.2 sanotaan, hahmontunnistuksen ja suodattamisen ero on monesti häilyvä.

Mielestäni suurin ongelma digitoidussa videokuvassa on parillisten ja parittomien vaakaviivojen väliset vaihe-erot. Televisiotekniikassa koko kuvaa ei muodosteta kerralla, vaan lomitetusti kahdella pyyhkäisyllä siten, että ensin piirretään parittomat vaakaviivat ja sitten parilliset. Luonnollisesti tästä seuraa se, että koska myös videokuva tallennetaan vastaavalla menetelmällä, nopeasti liikkuvat hahmot kuvassa näkyvät eri kohdissa parittomilla ja parillisilla vaakaviivoilla. PAL-kuvaa päivitetään 25 kertaa sekunnissa, joten yhdelle pyyhkäisylle jää aikaa 1 / 50 s. Näin ollen jos pelaaja juoksee esimerkiksi 7 m / s nopeudella, on kahden peräkkäisen vaakaviivan ero pelaajan osalta 14 cm. Televisiota katsellessa tätä ei huomaa, koska kuvaputken fosfori on sen verran hidasta, että edellinen kuva ei ehdi häipyä, kun uutta jo piirretään. Tapahtuu siis eräänlainen analoginen pehmentävä suodatus. Yksittäisessä kuvassa ero on kuitenkin merkittävä. Kuva 3.3 demonstroi tätä. Kohdassa a on pieni osa digitoitua videokuvaa. Parillisten ja parittomien vaakaviivojen ero on selkeästi havaittavissa. Kohdassa b on sama kuva, mutta siitä on poistettu parittomat vaakaviivat ja korvattu parillisilla. Tätä operaatiota kutsutaan *lomituksen poistoksi* (de-interlace). Pystytarkkuus luonnollisesti puolittuu, mutta kuten voidaan havaita, *todellinen* (havaittu) tarkkuus parantuu. Choi *et al.* [1997] ovatkin käyttäneet vain parillisia vaakaviivoja analysoidessaan videota.

(a) alkuperäinen kuva

(b) Sama kuva, kun parilliset vaakaviivat on poistettu ja korvattu parittomilla.

Kuva 3.3. Videokuvan parittomien ja parillisten viivojen eroavaisuus nopeassa liikkeessä.

VideoAnalyzerissä ei käytetä esiprosessointia. Parillisten ja parittomien vaakaviivojen ongelma ratkeaa sillä, että analyysissä käytetty resoluutio on 352 x 288 pikseliä, eli puolet PAL-kuvan resoluutiosta, joten videota digitoitaessa viivojen väliset erot interpoloidaan skaalauksen yhteydessä. Näin saatu kuva on parempi kuin kuvan 3.3 kohdassa a, mutta huonompi kuin kohdassa b. Yksi jatkokehityksen kohde onkin videon esiprosessointi VideoAnalyzerin sisällä. Kuva 3.4 demonstroi skaalauksen vaikutusta vaakaviivojen erovaisuuksiin. Kohdassa a on täysikokoinen PAL-kuva (720 x 576 pikseliä) puoleen kokoon skaalattuna. Kuva ei ole kovin terävä, sillä interpoloitaessa sekä parillisten että parittomien vaakaviivojen muodostamat kuvat jäävät kummittelemaan haamukuvina. Kohdassa b on sama kuva vastaavasti skaalattuna, mutta kuvasta on ennen skaalausoperaatiota poistettu parittomat vaakaviivat. Lomituksen poistolla tarkkuutta voitaisiin siis vielä lisätä.

(a) Täysikokoinen PAL-kuva puoleen kokoon skaalattuna.

(b) Sama kuva vastaavasti skaalattuna, mutta josta on ennen skaalausoperaatiota poistettu parittomat vaakaviivat.

Kuva 3.4. Lomituksen poiston vaikutus kuvan skaalauksessa.

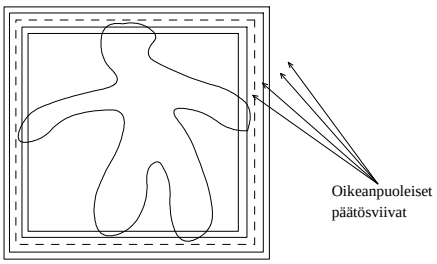
3.1.2. Klusterointi

Kuten johdannossa mainitaan, klusterointi on oleellinen vaihe automaattisessa videoanalyysissä, sillä sen tulokset vaikuttavat kaikkiin seuraaviin vaiheisiin. Tämän vuoksi VideoAnalyzerissä onkin testattu useita eri klusterointimenetelmiä. Ensimmäinen menetelmä on *kehysmalli* (frames model), jossa seurattava objekti näytetään systeemille piirtämällä laatikko (*bounding box*) sen ympärille. Objektia seurataan sitten tunnustelemalla laatikon ympäryksen muutoksia. Tämä tapahtuu laskemalla differenssimatriisi laatikon ympärykselle kahden peräkkäisen ruudun välille. Toinen menetelmä on klusterimalli, jossa tarkastellaan koko kuvan differenssimatriisia. Klusterimallista on myös kehittyneempi versio, *klusteritaulukkomalli* (cluster table model), jossa objektien polkujen etsintä ei tapahdu samaan aikaan klusteroinnin kanssa. Edelleen tästä on kaksi eri versiota. Ensimmäinen versio kelpuuttaa pisteitä etsimällä suuria kontrastivaihteluita. Toinen versio laajentaa etsintää hyödyntämällä myös kentän väriä. Käytetyt klusterointimenetelmät perustuvat Siermalan [1999] ja Siermala *et al.*:n [2000] tutkimukseen.

Differenssimatriisi lasketaan kahden peräkkäisen ruudun välisestä erotuskuvasta. Differenssimatriisin piste $D(i, j) = 1$, jos ja vain jos erotuskuvan vastaava piste ylittää raja-arvon r, joka on kokeellisesti haettu raja värimuutokselle. Muussa tapauksessa $D(i, j) = 0$. Varsinaisessa toteutuksessa differenssimatriisia ei tarvitse välttämättä laskea, vaan pisteet voidaan tarkistaa ajon aikana. Differenssimatriisia käytetään tässä pelkästään algoritmien selkeyttämiseksi.

Kehysmalli

Kehysmallin tarkoituksena on sulkea seurattava hahmo sisäkkäisiin kehyksiin. Kuvasta tarkastellaan värimuutoksia kehysten kohdalta ja korjataan kehysten sijaintia muutosten perusteella. Kehykset muodostetaan objektin ympärille siten, että käyttäjä määrittää aluksi seurattavan objektin piirtämällä sen ympärille kehyksen. Tämä on keskimmäinen katkoviivainen suorakaide kuvassa 3.5. Käyttäjän määrittelemän kehyksen sisä- ja ulkopuolelle muodostetaan säännöllisin välimatkoin lisää päätöskehyksiä. Päätöskehysten lukumäärä riippuu sovellusalueesta; seurattavien hahmojen nopeus vaikuttaa tarvittavien kehysten määrään. VideoAnalyzerissä on päädytty viiden päätöskehyksen käyttöön [Siermala *et al.*, 2000].



Kuva 3.5. Alustettu kehys ja päätöskehikko. [Siermala *et al.*, 2000]

Päätöskehikosta erotellaan oikean- ja vasemmanpuoleiset sekä ylä- ja alapäätösviivat. Kehysalgoritmi laskee differenssimatriisin kehikon viivoille ja tarkastelee, mitkä näistä ovat muuttuneet. Jos esimerkiksi ainoastaan objektia lähinnä oleva oikeanpuoleinen päätösviiva on muuttunut, siirretään keskimmäinen oikeanpuoleinen päätösviiva muuttuneen viivan kohdalle ja lasketaan paikat muille päätösviivoille. Tarkoituksena on siis pitää seurattavan hahmon ulommaisia reunoja keskimmäisen päätöskehyksen kohdalla. Kehysalgoritmi on esitetty pseudokoodina algoritmissa 3.1.

Eri suuntien päätösviivat osallistuvat päätöksentekoon itsenäisesti, mikä mahdollistaa kehikon kasvamisen ja mukautumisen seurattavan hahmon muotoon. Kehikon liiallista muutosta voidaan rajata esimerkiksi sisimmän kehikon pinta-alan avulla. [Siermala *et al.*, 2000].

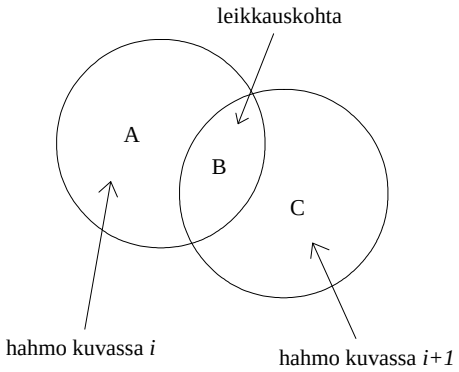
Algoritmi 3.1. Kehysalgoritmi (yhdelle objektille) [Siermala *et al.*, 2000]

```
1: i = 1
2: Alusta kehys ja laske päätöskehykset.
3: while ruutuja videosekvenssissä jäljellä do
4:     Laske differenssimatriisi D ruutujen i ja i+1 välille.
5:     for jokaiselle päätösalueelle (vasen, ylä, oikea ja ala) do
6:         if päätösalueen jonkin viivan kohdalla matriisissa D on kynnyksen ylittävä
           määrä ykkösiä then
7:             Siirrä keskimmäinen viiva muuttuneen viivan kohdalle ja laske paikat
               muille päätösviivoille
8:         end if
9:     end for
10:    i = i + 1
11: end while
```

Klusterimalli

Toinen tapa käyttää differenssimatriisia on laskea erotuskuva koko kuvassa näkyvästä jalkapallokentästä ja klusteroida löydetyt pisteet. Tarkasteltava alue on nyt paljon suurempi kuin kehysmallin tapauksessa – klusterimalli onkin huomattavasti hitaampi menetelmä kuin kehysmalli. Klusterointialgoritmi (esitetty pseudokoodina algoritmissa 3.2) käy differenssimatriisin läpi piste pisteeltä ja lisää sinne kelpuutetut pisteet listaan *L*, mikäli kuvan vastaavien pisteiden väriarvot eroavat taustasta. Tästä vertailusta kerron tarkemmin seuraavassa kappaleessa. Tämän jälkeen klusterointialgoritmi kutsuu pistealgoritmia, joka yhdistää pisteet, jotka ovat tietyn etäisyyden *r* sisällä toisistaan. Löydetyt klusterit ovat todennäköisesti hahmojen (pelaajien) synnyttämiä, joten

klusterit käydään läpi ja lähin edellisen ruudun klusteri liitetään aina löydettyyn klusteriin, jos alueella ei ole muita mahdollisia klustereita.



Kuva 3.6. Hahmon liikkuminen peräkkäisissä ruuduissa. [Siermala, 1999]

Hahmon liikkuessa kuvassa *i* ja *i+1*, sen klusterointialgoritmin määrittelemä sijainti on riippuvainen nopeudesta. Jos nopeus on pieni, hahmo liikkuu olemattomin nykäyksin ja hahmon uuden ja vanhan sijainnin leikkausalue on suuri. Värien muutos alueella saattaa olla myös vähäistä. Jos hahmo liikkuu nopeasti, se saattaa hypätä kahden ruudun välillä useita itsensä pituisia matkoja. Tällöin hahmo saatetaan kadottaa. Differenssimatriisista on siis löydettävissä aktiivisia alueita kohdasta, josta hahmo on poistunut ruudussa *i*, ja kohdasta, minne se on ilmestynyt ruudussa *i+1*. Kuvassa 3.6 nämä ovat kohdat A ja C. Jos hahmo on yksivärinen, positioiden leikkauskohtaan jää muuttumaton alue. Monivärisen hahmon tapauksessa myös alueella B saattaa ilmetä kynnyksen ylittäviä väriarvojen muutoksia. Klusterointialgoritmi pyrkii poistamaan pisteet, jotka syntyvät hahmon edellisen kuvan sijaintiin vertaamalla differenssimatriisiin kelpuutettua pistettä taustan väriin. Tarkoitus on siis poistaa kuvan 3.6 alue A. [Siermala, 1999]

Algoritmi 3.2. Klusterointialgoritmi (usealle objektille) [Siermala *et al.*, 2000]

1: $i = 1$
2: **while** ruutuja videosekvenssissä jäljellä **do**
3: Laske differenssimatriisi ruutujen *i* ja *i+1* välille.
4: Selaa differenssimatriisia järjestyksessä vasemmalta oikealle ja ylhäältä alas. Jos selaus löytää ykkösen ja pikselin väri poikkeaa taustan väristä oleellisesti, asetetaan luvun koordinaatit listaan *L*.
5: Klusterilista *C* = Pistealgoritmi(*L*)
6: Kiinnitetään kukin ryväs lähimpään edellisen ruudun hahmoon.
7: $i = i + 1$
8: **end while**

Algoritmi 3.3. Pistealgoritmi (saa parametrina listan *L*) [Siermala *et al.*, 2000]

1: Alusta lista *Clusters* tyhjäksi
2: **while** *L*:ssä alkioita **do**
3: Alusta vertailuvektori *V* tyhjäksi
4: Ota seuraava alkio listasta *L* ja aseta listan *V* ensimmäiseksi; poista alkio *L*:stä
5: **while** *V*:ssä alkioita **do**
6: Ota seuraava alkio *a* listasta *V*
7: **for** kaikille pisteille *b* listassa *L* **do**
8: **if** *a*:n ja *b*:n etäisyys on pienempi kuin *r* **then**
9: aseta *b* listan *V* viimeiseksi ja poista *L*:stä
10: **end if**
11: **end for**
12: **end while**
13: Lista *V* sisältää nyt löytyneen klusterin, jonka positio on *min j* (*j* $\hat{I}$ *V*)
14: Laita *min j* listaan *Clusters*
15: **end while**
16: Palauta lista *Clusters*

Klusteritaulukkomalli, versio 1

Klusteritaulukkomalli eroaa klusterimallista siten, että nyt polkuja ei etsitä heti klusteroinnin yhteydessä. Yksi syy tähän on se, että näin analyysin laskennallista taakkaa voidaan vähentää. Polkujen etsintä pitää myös tietysti suorittaa, mutta se voidaan tehdä analyysin jälkeen. Klusteroinnin tehokkuus on olennainen tekijä, mikäli analyysi halutaan tehdä reaaliaikaisesti videolta videodigitointikortin avulla. Klusteritaulukkoalgoritmi käy videon läpi ja tallentaa löydettyjen klustereiden sijainnit taulukkoon. Klusteritaulukkomalli ei hyödynnä differenssimatriisia, vaan pisteiden kelpuutus tapahtuu vertaamalla vierekkäisten pikseleiden väriarvoja. Hahmojen rajat ovat oletettavasti teräviä värimuutoksia vierekkäisissä pikseleissä. Olettamalla, että kenttä on lähes tasavärinen, voidaan tätä käyttää hyväksi hahmon etsinnässä. Menetelmässä kuvaa selataan järjestyksessä vasemmalta oikealle ja ylhäältä alas (vain kentän alueelta). Jokaisen pisteen kohdalla verrataan pikselin väriarvoja seuraavan pikselin väriarvoihin. Väriarvojen poikkeama *h* pisteelle *(i, j)* määritellään

$$h(i, j) = sum(dR((i, j),(i+1, j)), dG((i, j),(i+1, j)), dB((i, j),(i+1, j))),$$

missä *dR(a, b), dG(a, b) ja dB(a, b)* ovat värikomponenttien erotukset kohdissa *a* ja *b*. Jos väriarvot ylittävät kokeellisesti haetun kynnyksen *t,* eli *h(i, j)* > *t,* on pikseli luultavasti hahmon rajalla. Menetelmällä löydetyt pisteet yhdistetään käyttämällä klusterimallin mukaista pistealgoritmia.

Klusteritaulukkomalli, versio 2

Klusteritaulukkomallin ensimmäisessä versiossa on useita puutteita. Ensinnäkään se ei ota taustan väriä huomioon. Se on myös erittäin hidas ja kuluttaa paljon muistia ja tehoa. Muisti ei tahdo riittää millään, kun kaikki klusteridata tallennetaan käyttömuistiin. Resursseihin liittyvistä ongelmista on kerrottu tietorakenteita käsittelevässä alaluvussa. Klusteritaulukkomallin toinen versio perustuu alueen kasvattamiseen. Algoritmi 3.4 käy kuvaa läpi piste pisteeltä, ja jos löytyy sopiva pikseli, kutsutaan funktiota EtsiPelaaja. Se tarkistaa, onko pisteessä jo käyty, ja onko klusterin koko suurempi kuin *maxpix*. Koska on käytännössä mahdotonta, että hahmo olisi kooltaan esimerkiksi yli 1000 pikseliä, voidaan kokotarkistuksella vähentää virhetulkintoja. Ennen kaikkea sillä voidaan estää pinon ylivuotoja. Jos hahmo ei ole liian suuri ja pisteessä ei ole vielä käyty, kutsutaan funktiota PisteTarkistukset, joka tarkistaa onko piste kelvollinen. Jos piste on kelvollinen, funktio EtsiPelaaja etsii pisteen ympäriltä rekursiivisesti lisää sopivia pisteitä. Etsittävän alueen koko on sekä vaaka että pystysuunnassa ennalta määritelty *–ScanR ... ScanR*. Kun sopivia pisteitä ei enää löydy, lasketaan muodostuneen klusterin pisteistä klusterin keskimääräinen väri, keskipiste, leveys ja koko (pikseleissä). Mikäli klusteri on suurempi kuin *minpix*, sen tiedot tallennetaan tiedostoon. Minimikokotarkistuksella voidaan karsia klustereita, jotka ovat niin pieniä, että ne luultavimmin eivät ole pelaajia. Tämän jälkeen jatketaan kuvan normaalia läpikäyntiä. Kaikki läpikäydyt pisteet merkitään käsitellyiksi, jotta ei jouduttaisi loputtomiin silmukoihin tai yleensäkään jouduttaisi tekemään turhia laskutoimituksia.

Klusteritaulukkomallin toinen versio käyttää pikseleiden kelpuuttamiseen mallin sovittamista ja reunojen tunnistusta. Käyttäjä antaa aluksi esimerkin taustasta piirtämällä laatikon johonkin taustan kohtaan – mieluiten sellaiseen kohtaan, jossa mahdollisimman suuri skaala taustan sävyistä on näkyvissä. Tästä lasketaan taustan värikomponenttien keskiarvot ja luottamusvälit, joita käytetään klusterointiin. Mikäli satunnaismuuttujan $\underline{x}$ arvo poikkeaa odotusarvostaan m korkeintaan 1,96 kertaa keskihajontansa s verran, eli jos $|\underline{x}-m| \leq 1{,}96s$, niin tällöin poikkeama ei normaalijakauman mukaan ole tilastollisesti merkitsevä. Tälle alueelle sattuu 95 % $\underline{x}$:n arvoista [MAOL, 1992]. Etsitään siis pisteitä, jotka eivät osu tälle välille. Värikomponentin keskiarvo *iAve* lasketaan jakamalla kenttäesimerkistä löydettyjen värikomponentin esiintymien summa kenttäesimerkin pistemäärällä *n*. Värikomponentin keskihajonta *iDev* lasketaan kaavalla

$$iDev = \sqrt{\frac{\sum_{i=1}^{n} (x_i - iAve)^2}{n}}.$$

Värikomponentin 95 %:n luottamusväli *iLimit* on 1.96 * iDev.

Funktio PisteTarkistukset vertaa kyseessä olevaa pistettä taustaan. Jos piste eroaa tarpeeksi taustasta, eli $|\underline{x}-m| > 1{,}96s$, se kelpuutetaan. Pistettä verrataan myös viereiseen pisteeseen, kuten klusteritaulukkomallin ensimmäisessä versiossa. Tässä versiossa kuitenkin tarkastellaan yksittäisiä värikomponentteja niiden summan sijasta.

Algoritmi 3.4. Uusi klusterointialgoritmi

```
1: while ruutuja videosekvenssissä jäljellä do
2:     for kaikille ruudun pisteille(x, y) do
3:         XMin = XMax = YMin = YMax = -1
4:         iaRed = iaGreen = iaBlue = pixels = 0
5:         EtsiPelaaja(x, y)
6:         if pixels ≥ minpix then
7:             s.RAvg = iaRed / pixels
8:             s.GAvg = iaGreen / pixels
9:             s.BAvg = iaBlue / pixels
10:            s.Width = XMax – XMin
11:            s.x = (s.Width / 2) + XMin
12:            s.y = YMax
13:            s.size = pixels
14:            Kirjoita klusteri s tiedostoon
15:        end if
16:    end for
17: end while
```

Algoritmi 3.5. EtsiPelaaja (*x*, *y*)

```
1: if pisteessä (x, y) jo käyty tai pixels > maxpix  then return
2: Merkitse piste (x, y) läpikäydyksi.
3: if PisteTarkistukset(x, y) = 1  then
4:     pixels = pixels + 1
5:     if YMin = -1 then
6:         YMin = YMax = y
7:         XMin = XMax = x
8:     else
9:         if YMin > y  then YMin = y else if YMax < y then YMax = y
10:        if XMin > x then XMin = x else if XMax < x then XMax = x
11:    end if
12:    for iy = -ScanR to ScanR do
13:        for ix = -ScanR to ScanR do
14:            if ix = 0 ja iy = 0 then continue
```

15: **if** pisteessä (*x+ix, y+iy*) ei ole vielä käyty **then**
16: EtsiPelaaja(*x+ix, y+iy*)
17: **end if**
18: **end for**
19: **end for**
20: **end if**

Algoritmi 3.6. PisteTarkistukset (*x, y*)

1: *rc = 0*
2: *iRed, iGreen, iBlue* = Kuvan värikomponentit pisteessä (*x, y*)
3: *iRAve, iGAve, iBAve* = Kentän värikomponenttien keskiarvot
4: *iRLimit, iGLimit, iBLimit* = Luotettavuusvälit taustan värikomponenteille
5: **if** (abs(*iRed – iRAve*) > *iRLimit*) tai
(abs(*iGreen – iGAve*) > *iGLimit*) tai
(abs(*iBlue – iBAve*) > *iBLimit*) **then** *rc* = 1
6: *i2Red, i2Green, i2Blue* = Kuvan värikomponentit pisteessä (*x+1, y*)
7: **if** (abs(*iRed – i2Red*) > *threshold*) tai
(abs(*iGreen – i2Green)* > *threshold*) tai
(abs(*iBlue – i2Blue*) > *threshold*) **then** *rc* = 1
8: **if** *rc* = 1 **then**
9: *iaRed = iaRed + iRed*
10: *iaGreen = iaGreen + iGreen*
11: *iaBlue = iaBlue + iBlue*
12: **end if**
13: Palauta *rc*

3.1.3. Polkujen etsintä

Klusterointivaiheen jälkeen voidaan etsiä hahmojen yhtenäisiä liikeratoja (polkuja). Poluista pyritään löytämään ainakin pisimmät ja yksiselitteisimmät. Tällöin ainakin toisista etäällä olevien pelaajien polkujen tulisi löytyä. Löydetyt hahmot on klusterointivaiheessa asetettu taulukkoon, jossa sarakkeet edustavat yhden ruudun hahmoja ja rivit ajan hetkiä. Tämä alaluku kuvaa kuinka polkujen etsintä suoritetaan klusteritaulukkomallin versioissa yksi ja kaksi. VideoAnalyzerin aikaisemmat toteutukset suorittivat polkujen etsinnän orgaanisena osana klusterointia ja niiden käyttämät menetelmät onkin esitelty jo aikaisemmin. Polkujen etsintä perustuu Siermalan [1999] ja Siermala *et al.*:n [2000] tutkimukseen.

Alun perin polkujen etsintään käytettiin algoritmia, joka edellyttää, että polun eri ajan hetkien havaintojen täytyy olla ristiriidattomia. Mikäli kaksi tai useampia hahmoja löytyy säteen *r* sisältä seuraavasta ruudusta, polun etsintä lopetetaan siihen. Säde *r* on

siis ennalta määritelty raja-arvo objektin kahden peräkkäisen ruudun esiintymien maksimietäisyydelle. Alkuperäinen Taulukko_algoritmi on esitetty pseudokoodina algoritmissa 3.7. Se käy taulukon läpi ja etsii kaikki solut, joissa on klustereita. Tämän jälkeen se kutsuu funktiota Rekursiivinen_algoritmi, jolle se antaa solun sijainnin parametreina. Rekursiivinen_algoritmi etsii koko pelaajan kulkeman polun ja pysähtyy, kun se tunnistaa tilanteita, joissa kaksi tai useampia hahmoja on säteen *r* sisällä. Mikäli löydetty polku on pidempi kuin ennalta määritelty raja-arvo *l*, se hyväksytään poluksi.

Nykyään polkujen etsintä perustuu lähimmän naapurin menetelmään. Nyt useampia hahmoja voi olla säteen *r* sisällä. Tästä ei välttämättä ole pelkästään hyötyä, sillä liian lähellä toisiaan olevat hahmot sekoittuvat helposti. Polut haetaan etsimällä kuvan (sarakkeen) kutakin hahmoa lähinnä oleva hahmo seuraavasta kuvasta. Eli valitaan klusteri ruudusta *k* ja etsitään seuraavasta ruudusta *k+1* sen sijaintia lähinnä oleva klusteri. Tämä tarkistus tehdään myös toiseen suuntaan. Ruudun *k* valitun klusterin täytyy olla myös lähinnä ruudusta *k+1* löydettyä klusteria. Näin pyritään välttämään havainnon ”varastaminen” toiselta polulta, josta oli puhetta luvussa 2 yhteisen naapurin etäisyyden yhteydessä. Lisäksi havaintojen täytyy olla tietyn raja-arvoetäisyyden sisällä. Tämä sen vuoksi, että jos ruudusta *k+1* löydetään hetkellisesti esimerkiksi vain yksi klusteri, joka on eri puolella kenttää kuin ruudun *k* käsiteltävänä oleva klusteri, niitä ei virheellisesti yhdistetä – vaikka klusterit ovatkin aidosti lähimpinä toisiaan. Taulukko_algoritmi on nykyisessä toteutuksessa sama kuin ennen, mutta Rekursiivinen_algoritmi on kirjoitettu uudelleen. Vaikka nimi on pidetty samana, se ei itse asiassa ole enää rekursiivinen. Uusi versio on esitelty pseudokoodina algoritmissa 3.9. Funktiossa käytetty vakio *frames* on koko videon sisältämien ruutujen määrä. Funktiossa Etsi_Lähin käytetty vakio *r* on sama maksimietäisyys kahden havainnon välillä kuin vanhassa Rekursiivinen_algoritmi –funktiossa.

Algoritmi 3.7. Taulukko_algoritmi (*r*, *l*) [Siermala *et al.*, 2000]

```
1: Alusta lista path_lis
2: for kaikille riveille i (ruuduille) do
3:     for kaikille sarakkeille j, joissa on klusteri do
4:         path = Rekursiivinen_algoritmi (i, j, r)
5:         if pituus(path) ≥ l then
6:             Laita path listaan path_list
7:         end if
8:     end for
9: end for
```

Algoritmi 3.8. Rekursiivinen_algoritmi (*i*, *j*, *r*) [Siermala *et al.*, 2000]

```
1: counter = 0
```

2: Alusta tyhjä lista *path_points*
3: **for** kaikille klustereille *k* ruudussa (*i+1)* **do**
4: **if** etäisyys(piste(*j, i*), piste(*k, i+1*)) ≤ *r* **then**
5: *counter* = *counter* + 1
6: *m* = *k*
7: **end if**
8: **end for**
9: **if** *counter* = 1 **then**
10: *counter* = 0
11: **for** kaikille klustereille *l* ruudussa *i* **do**
12: **if** etäisyys(piste(*l, i*), piste(*m, i+1*)) ≤ *r* **then**
13: *counter* = *counter* + 1
14: *n* = *l*
15: **end if**
16: **end for**
17: **end if**
18: **if** *counter* = 1 ja *m* = *n* **then**
19: *path_points* = Rekursiivinen_algoritmi (*i+1*, *m*, *r*)
20: Lisää *i* ja *j* listaan *path_points*
21: Tyhjää solu(*j*, *i*)
22: **end if**
23: Palauta *path_points*

Algoritmi 3.9. Rekursiivinen_algoritmi (*i*, *j*, *r*)

1: *o* = *i; found* = 1
2: Alusta tyhjä lista *path_points*
3: **while** *o* < *frames* ja *found* = 1 **do**
4: *min21* = Etsi_Lähin(*j*, *o*, o+*1*)
5: **if** *min21* = -1 **then**
6: *found* = 0
7: **end if**
8: *min12* = Etsi_Lähin(*j*, *o+1*, o)
9: **if** *min21* = *min12* **then**
10: Laita *o+1* ja *min21* listaan *path_points*
11: Tyhjää solu(*min21*, *o+1*)
12: **else**
13: *found* = 0
14: **end if**
15: *o* = *o* + 1
16: **end while**

17: Palauta *path_points*

Algoritmi 3.10. Etsi_Lähin (*i*, *cur*, *cmp*)

```
1: count = 0; closest = r
2: for kaikille klustereille k ruudussa cmp do
3:     dist = etaisyys(piste(i, cur), piste(k, cmp))
4:     if dist < r ja dist < closest then
5:         index = i
6:         closest = dist
7:         count = count + 1
8:     end if
9: end for
10: if count = 0 then
11:     Palauta –1
12: else
13:     Palauta index
14: end if
```

Polkujen etsintää sotkevat klusteroinnissa tapahtuneet virhehavainnot. Eri ruuduista saattaa löytyä eri lukumäärä objekteja, mikä aiheuttaa hankaluuksia päättelyssä. Joku polku saattaa jäädä ilman klusteria. Virheitä voi myös syntyä, jos jossakin kuvassa klustereita ei löydy lainkaan. Tällöin on vaarana, että polku katkeaa. Virhe tapahtuu myös, jos pelaaja kaatuu tai hyppää. Perspektiivistä johtuen ohjelma voi erehtyä tulkitsemaan, että pelaaja liikkuu yhtäkkiä useita metrejä.

Alun perin etäisyystarkastelut suoritettiin taulukkomalleissa pelkästään bittikarttojen koordinaateilla, joten objektien etäisyyttä kameraan ei tarkasteluissa otettu huomioon. Koska samoja etäisyysmittojen raja-arvoja käytettiin lähelle ja kauas, kauempana oleva pelaaja saattoi näyttää liikkuvan yhtäkkiä (sekunnin murto-osassa) esimerkiksi 20 metriä, vaikka olikin kyse siitä, että pelaaja havaittiin pikseli tai pari ylempää kuin edellisessä kuvassa. Kameran lähellä liikkuvien pelaajien kohdalla tästä ei tietenkään aiheutunut ongelmia.

Olen pyrkinyt korjaamaan polun etsinnän ogelmia sillä, että löydetyt polut käydään läpi ja niihin tehdään erilaisia tarkistuksia. Ensinnäkin niihin tehdään etäisyystarkistukset myös kentän koordinaatistossa. Määritellään maksimimatka *lmax*, jonka pelaaja voi liikkua yhden ruudun aikana. Voidaan olettaa, että pelaaja ei voi liikkua nopeammin kuin 10 metriä sekunnissa. Tämäkin olisi jo huippujuoksijan nopeus. Kentän koordinaatistossa tapahtuva etäisyysvertailu tietylle havainnolle tehdään kahden peräkkäisen havainnon välillä. Siten on mahdollista, että jos polun ensimmäinen

havainto on virheellinen, mitään seuraavista havainnoista ei kelpuuteta. Tämän vuoksi kehitin systeemin, joka asettaa polun alun aina seuraavaan ruudun havaintoon ja käy polun uudestaan läpi, mikäli tulee määrätty määrä peräkkäisiä virhehavaintoja. Algoritmissa tätä kuvaa ennalta määrätty vakio *maxerr*. Yksittäisiä etäisyyden *lmax* ylittäviä virhehavaintoja ei oteta mukaan polkuun. Algoritmissa 3.11 esitetty Heuristiikka_algoritmi ottaa parametrinaan polun ja palauttaa sen suodatetun version. Kentän koordinaatistossa tapahtuvan tarkastelun merkityksen voi nähdä kuvasta 3.7. Suodattamattoman kuvan harmaat viivat kuvaavat virheitä, jotka syntyvät, jos etäisyysvertailut tehdään pelkästään bittikarttojen koordinaateilla. Karkeasti voidaan arvioida, että kameraan nähtynä keskiviivan jälkeen pelaajat ovat niin pieniä, että havaintojen taso laskee huomattavasti. Tämä on havaittavissa myös kuvasta 3.7.

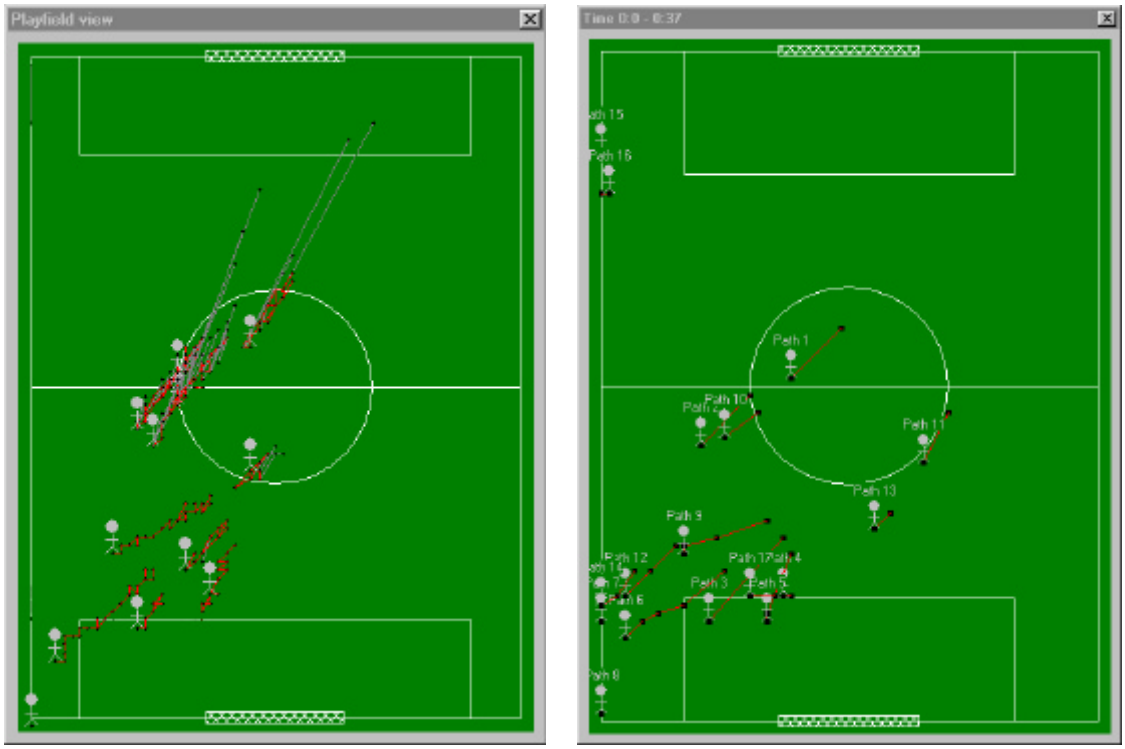
Algoritmi 3.11. Heuristiikka_algoritmi (*V*)

```
1: Alusta tyhjä lista tmp
2: istart = 0; i = istart
3: repeat
4:     if etäisyys(V[i], V[i+1]) ≤ lmax then
5:         Lisää piste V[i] listaan tmp
6:         ierror = 0
7:     else
8:         ierror = ierror + 1
9:         if ierror ³ maxerr then
10:            Tyhjää lista tmp
11:            ierror = 0; i = istart; istart = istart + 1
12:        end if
13:    end if
14:    i = i + 1
15: until i ≥ pituus(V)
16: Palauta lista tmp
```

Yritin myös päätellä objektien päällekkäisyyttä klusterien äkillisellä kasvamisella. Mikäli jonkun polun klusterin koko kasvoi yhden ruudun aikana 1.5-kertaiseksi, se merkittiin ”jaetuksi”. Vastaavasti, mikäli koko pienentyi äkillisesti puoleen, ”jaettu”-ominaisuus poistettiin. Näin muut polut pystyivät hyödyntämään jaettuja klustereita. Tällä oli kuitenkin taipumusta aiheuttaa loputtomia silmukoita. Hylkäsin ajatuksen, koska pelkän koon hyödyntäminen päällekkäisyyksien päättelyyn tuntui liian epäluotettavalta.

Kun kelvollinen polku on löytynyt, siitä karsitaan pois liikoja pisteitä. Koska tunnistuksessa tapahtuu koko ajan pientä epätarkkuutta, polkuihin syntyy helposti

sahaliikettä. Suurimmat virheethän poluista on jo karsittu edellä mainituilla tavoilla, mutta pienestä sahaliikkeestä on mahdotonta sanoa, milloin on kyseessä oikea ja milloin väärä havainto. Liikapisteiden karsimiseen kokeilin kahta eri tapaa. Ensimmäinen tapa on karsia pois esimerkiksi neljä pistettä viidestä. Selvästikään tämä tapa ei ole paras mahdollinen. On edelleen aivan mahdollista, että syntyy pistesumppuja. Näin tapahtuu, jos pelaaja seisoo paikallaan yli viiden ruudun ajan. Koska tarkoituksena on saada aikaan janoja, olisi mielekästä, että janat olisivat ainakin tietyn pituisia. Toinen tapa onkin karsia pisteitä, jos etäisyys edellisestä kelpuutetusta pisteestä on liian pieni. Kuvassa 3.7. on esitetty polkujen etsinnän suodattamaton ja suodatettu tulos. Liikapisteiden karsintaan on käytetty jälkimmäistä tapaa.



Kuva 3.7. Polkujen etsinnän tulos suodattamattomana ja suodatettuna.

3.2. Tietorakenteet

Tietorakenteiden suunnittelu on olennainen osa systeemiä, jossa informaation määrä on suuri. Yksi jalkapallopelin puoliaika kestää 45 minuuttia. Kun video on digitoitu puolella PAL-tarkkuudella, AVI-tiedosto vie pakattunakin yli gigatavun tilaa. Pakkaamattomana gigatavuun ei mahdu kuin pari minuuttia videota. Kiintolevyjen koot kasvavat räjähdysmäisesti ja yhä tehokkaampia pakkausmenetelmiä kehitellään jatkuvasti, joten ongelma ei ole kovin merkittävä ainakaan massamuistin kannalta tarkasteltuna. Analyysivaiheessa videota käsitellään purettuna, joten prosessoitavan datan määrä pysyy kuitenkin aina samana.

AVI-tiedoston maksimikoko on 2 Gt. Muun muassa tämän vuoksi olen kehittänyt systeemiä ottamaan huomioon reaaliaikaisen videon käsittelyn mahdollisuuden. Jotta systeemistä voitaisiin tehdä reaaliaikainen, tietorakenteiden täytyy olla mahdollisimman tehokkaita. Alunperin VideoAnalyzerin tietorakenteet olivat dynaamisia linkitettyjä listoja. Tästä oli huomattavia haittoja. Ensinnäkin osoittimia oli aivan liikaa. Dataa on muutenkin paljon ja tietorakenteiden ylläpitoon tarvittavat osoittimet vievät lähes yhtä paljon tilaa kuin varsinainen tallennettava tieto. Toiseksi, epäsuora osoitus hidastaa prosessointia huomattavasti. Kolmanneksi, tietorakenteet eivät kuitenkaan mahdu kerralla muistiin. Nykyään ei tietenkään ole tavatonta törmätä tietokoneeseen, jossa on useampi gigatavu muistia, mutta koska tarkoituksena on kehittää mahdollisimman edullinen systeemi, olen pyrkinyt ratkaisuun, jossa kaikkea dataa ei talleteta kerralla muistiin.

Muistin tarve vähenee huomattavasti, kun klusteridata tallennetaan tiedostoon jo prosessointivaiheessa. Lisäksi dataa voidaan lukea levyltä sopivan kokoisissa palasissa, joten muistiongelman pitäisi hävitä kokonaan. Linkitettyjen listojen sijasta klustereiden tallentamiseen käytetään taulukoita. Voidaan olettaa, että löydettyjen klustereiden määrä on jokseenkin vakio eri ruutujen välillä, joten taulukoiden käyttö on muistin kannalta tehokkaampaa kuin linkitettyjen listojen. Lisäksi suora osoitus on huomattavasti nopeampaa kuin epäsuora osoitus. Polkudatan tietorakenteita en sinänsä muuttanut. Ne ovat edelleen linkitettyjä listoja. Polkudatasta säilytetään muistissa kuitenkin vain lopulliset suodatetut polut.

Yksi tapa vähentää käsiteltävän datan määrää on luonnollisesti vähentää käsiteltävien ruutujen määrää. Normaalisti video on tallennettu 25 *ruudun sekuntinopeudella* (frames per second (FPS)). Käsiteltävien kuvien määrää voidaan vähentää ainakin 15 ruutuun sekunissa ilman, että tulokset heikentyvät merkittävästi. Jopa 5 ruudun sekuntinopeus on yleensä riittävä. Tätä voidaan perustella sillä, että objektien paikantamisalgoritmi tunnistaa pelaajien sijainnin metrin tarkkuudella. Jos oletetaan, että ihminen pystyy juoksemaan enintään 10 metriä sekunnissa, 10 ruudun sekuntinopeuden pitäisi riittää nopeimmankin pikajuoksijan tunnistamiseen. Poikkeuksen tekee tietysti pallo, joka voi liikkua vielä huomattavasti nopeammin. Pallon seuraaminen on kuitenkin niin vaikeaa, ettei siihen ole VideoAnalyzerin kehittämisessä kiinnitetty sen suurempaa huomiota.

3.3. Objektien paikantaminen kentällä

Kolmiulotteinen liike näkyy kuvaruudulla kahdessa ulottuvuudessa. Kuvasta ei voida suoraan mitata pelaajien sijaintia, joten kuvan koordinaatit on muutettava kentän koordinaatteihin. Videokuvassa kolmiulotteinen maailma on projisoitu kameran linssin läpi filmin pinnalle; filmille syntyneet suhteet siirtyvät katseluvaiheessa kuvaruudulle.

Ongelmana on se, että emme voi tietää projektiotasona toimivan filmin sijaintia, emmekä kentän ja projektiotason välistä kulmaa. [Siermala, 1999]

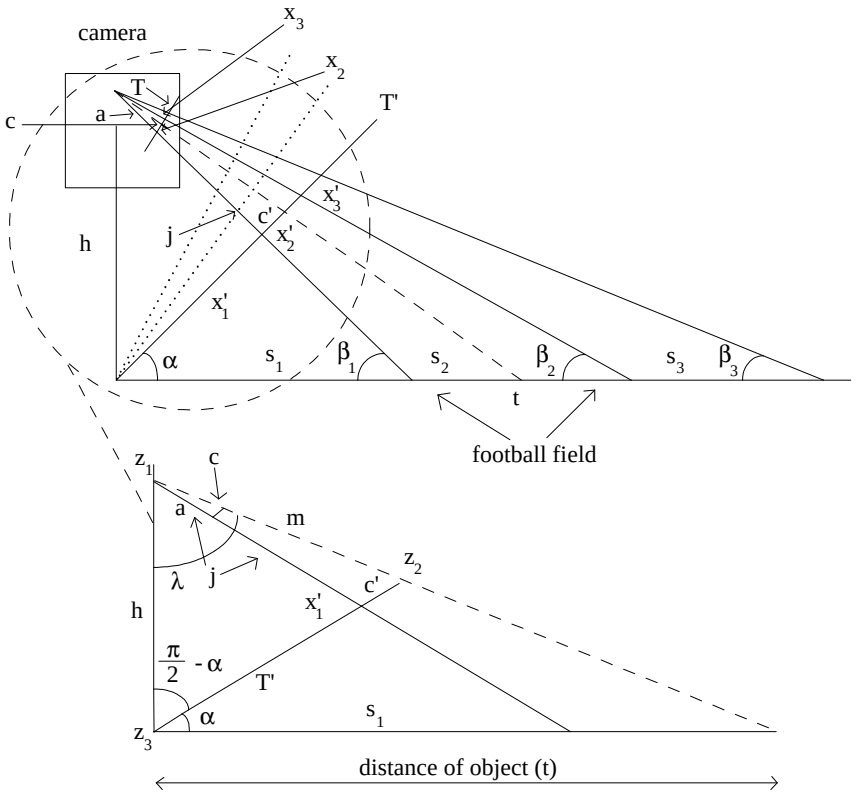
Objektien paikantaminen perustuu Siermala *et al.*:n [2000] tutkimukseen. Aihetta käsitteli alunperin Siermala [1999]. Tämän alaluvun tarkoituksena on tarkastella kysymystä, miten projektiotason kanssa menetellään ja miten kuvan koordinaatit siirretään kentälle.

Objektien paikantaminen perustuu kiintopisteiden käyttöön. Analyysin alussa käyttäjän määrittelemien kentän pisteiden ja kentän ominaisuuksien avulla perspektiivinen taso voidaan kääntää näytölle. Siermalan [1999] mukaan ongelmasta on erotettavissa kolme eri kokonaisuutta: pelaajien etäisyyden laskeminen, pelaajan kulman suhde kamerasta avautuvaan sektoriin ja polaaristen koordinaattien siirtäminen kentän koordinaatteihin. Etäisyyttä laskettaessa käytetään virtuaalista projektiotasoa *T'*, joka muodostetaan kameran kohdalle pelikentän korkeudelle. Kuvaruudulla olevat pisteet siirretään virtuaaliselle tasolle, jolloin kaikki pisteet filmin tasolla *T* ja virtuaalisella projektiotasolla *T'* ovat samassa suhteessa toisiinsa. Tason *T'* avustuksella tapahtumat voidaan siirtää kentän koordinaatteihin.

Aluksi tutustutaan *T'*:n laskemiseen. Tämän jälkeen kameran tarkastelusektoriin rakennetaan polaarinen koordinaatisto. Lopuksi polaarinen koordinaatisto siirretään pelikentän koordinatistoon yksinkertaista muunnoskaavaa käyttäen.

3.3.1. Etäisyys

Kahden objektin todellista etäisyyttä ei voi suoraan määrittää kuvasta. Laskutoimituksia varten tarvitaan seuraavia tietoja: kentän pituus *H* ja leveys *W*, kameran sijainti suhteessa kenttään kolmiulotteisessa avaruudessa ja kolme pistettä s_1, s_2 ja s_3 eri etäisyyksillä kamerasta. Kuvat 3.8 ja 3.9 esittävät tilanteen graafisesti. Yksi systeemin vaatimuksista on se, että kameran tulee olla yhdessä kentän nurkista. Tämä helpottaa laskutoimituksia.



Kuva 3.8. Kolme tunnettua pistettä, virtuaalinen projektiotaso *T'* ja muuttujia. [Siermala *et al.*, 2000]

Pääongelmana on kameran projektiotason kulman selvittäminen. Filmin pinnan ja kentän välistä kulmaa on lähes mahdotonta tietää. Ongelma on ratkaistu käyttämällä virtuaalista projektiotasoa *T'*. Aluksi taso *T'* asetetaan kameran juureen. Taso *T'* alustetaan käyttämällä hyväksi kentän tunnettuja pisteitä, jotka näkyvät kuvassa 3.9. Olkoon *h* kameran korkeus kentästä ja sen etäisyydet kolmeen tunnettuun pisteeseen s_1, s_1+s_2 ja $s_1+s_2+s_3$. Tällöin kulmat b_1, b_2 ja b_3 voidaan laskea kaavoilla

$$b_1 = \arctan(\frac{h}{s_1})$$

$$b_2 = \arctan(\frac{h}{s_1 + s_2})$$

$$b_3 = \arctan(\frac{h}{s_1 + s_2 + s_3}),$$

jolloin sinilauseen perusteella voidaan laskea seuraava suhde

$$\frac{\sin(b_1)}{x'_1} = \frac{\sin(p - (a + b_1))}{s_1}.$$

Ruudun alalaita on nyt pisteessä x'_1 virtuaalisella projektiotasolla *T'*. Yhtälön oikeanpuoleinen osoittaja voidaan sieventää muotoon $sin(a+b_1)$, jolloin

$$x'_1 = \frac{s_1 \sin(b_1)}{\sin(a + b_1)}.$$

Piste $x'_1+x'_2$ voidaan laskea kaavalla

$$x'_1+x'_2 = \frac{(s_1 + s_2)\sin(b_2)}{\sin(a + b_2)} \quad (1)$$

ja viimeinen piste $x'_1+x'_2+x'_3$ voidaan laskea kaavalla

$$x'_1+x'_2+x'_3 = \frac{(s_1 + s_2 + s_3)\sin(b_3)}{\sin(a + b_3)}. \quad (2)$$

Etäisyydet x'_2 ja x'_3 voidaan laskea käyttämällä yhtälöitä (1) ja (2). Nyt voidaan verrata kentän etummaisen ja taemmaisen puoliskon suhdetta kahdessa eri projektiotasossa *T* ja *T'*

$$\frac{x_3}{x_2} = \frac{x'_3}{x'_2}. \quad (3)$$

Kulmaa *a* muutetaan, kunnes suhde (3) on yhtäsuuri. Näin saadaan approksimaatio filmin kulmalle kuvaustilanteessa. Tason *T'* kulma on pelikentän suhteen sama kuin filmin pinnan kulma kameran sisällä. Filmin kulma approksimoidaan alustusvaiheessa.

Polttopisteen ja projektiotason välinen etäisyys *j* lasketaan kosinilauseella

$$j = \sqrt{x'^2_1+h^2 - 2*x'_1*h*\cos(\frac{p}{2} - a)}.$$

Jotta etäisyys *c* voidaan projisoida tasolle *T'* täytyy tietää etäisyys *a* ja se selvitetään kaavalla

$$a = \frac{x_2}{x'_2} * j.$$

Etäisyys *c'* tasolla *T'* lasketaan kaavalla

$$c' = \frac{j}{a} * c,$$

jolloin pisteiden z_1, z_2, z_3 määräämän kolmion sivu x'_1+c' on tunnettu. Nyt sivu *m* voidaan laskea kosinilauseella

$$m = \sqrt{h^2 + (x'_1+c')^2 - 2*h*(x'_1+c')*\cos(\frac{p}{2} - a)}.$$

Käyttämällä sinilausetta saadaan

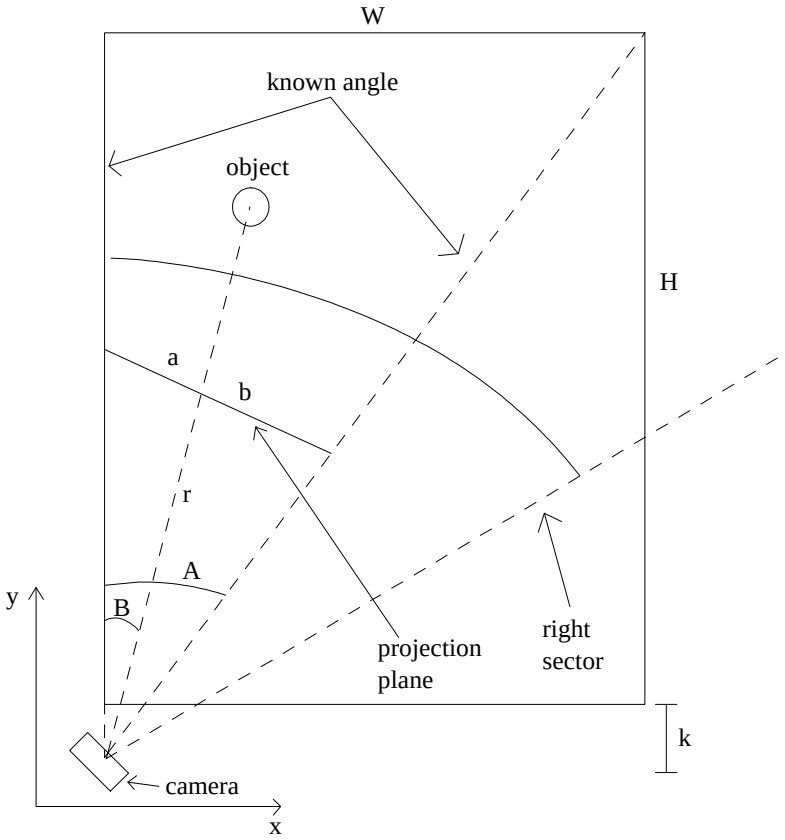
$$\frac{\sin(\frac{p}{2}-a)}{m}=\frac{\sin(l)}{x'_1+c'},$$

jolloin muuttuja l voidaan selvittää:

$$l=\arcsin(\frac{\sin(\frac{p}{2}-a)*x'_1+c'}{m}).$$

Kulman l avulla voidaan laskea tapahtuman *t* etäisyys kamerasta kaavalla

$$t=\tan(l)*h.$$



Kuva 3.9. Kameran sijainti ja sen ympärille muodostuva polaarikoordinaatisto. [Siermala *et al.*, 2000]

3.3.2. Suunta

Kuvasta 3.9 nähdään, että kulma *A* voidaan laskea likimääräisesti suorakulmaisen kolmion laskukaavalla

$$A\approx\arctan(\frac{W}{H+k}),$$

missä *W* on kentän leveys, *H* on kentäs pituus ja *k* on etäisyys kameran ja kentän alun välillä. Mittaamalla kahden vastakkaisen päädyn välisen etäisyyden, saadaan etäisyys

$a+b$. Mittaamalla tapahtuman etäisyys a kentän vasemmasta reunasta voidaan kulma B laskea likimääräisesti

$$B \approx \frac{a}{a+b} A.$$

3.3.3. Kentän koordinaatistoon siirtyminen

Edellä kerrottiin, kuinka tapahtuman etäisyys ja suunta lasketaan. Saadut tulokset ovat polaarikoordinaateissa. Ne muunnetaan suorakulmaisiin koordinaatteihin seuraavilla kaavoilla:

$$y = r\cos(B)$$
$$x = r\sin(B).$$

Nyt kamera on origossa ja x ja y ovat etäisyyksiä siitä. Lopuksi origo pitää muuttaa kentän lähimpään nurkkaan.

4. Tutkimustuloksia

Jalkapallovideon automaattinen analysointi osoittautui erittäin hankalaksi tehtäväksi. Yhden paikallaan olevan kameran vaatimus on todella ankara, kun ottaa huomioon, kuinka suurta aluetta on tarkoitus seurata. Algoritmeja ja heuristiikkoja kehittämällä analyysin tarkkuutta voidaan kuitenkin parantaa huomattavasti. Algoritmien ja tietorakenteiden optimoinnilla analyysiä voidaan vastaavasti huomattavasti nopeuttaa.

Tässä luvussa kerron reaaliaikaisuuden analyysille asettamista rajotteista. Lisäksi vertailen eri tekniikoita ja kerron, minkälaisia tuloksia niillä on saatu aikaan. Pohdin myös keskeisintä tutkimusongelmaa eli hahmojen päällekkäisyyden hallintaa. Lopuksi arvioin suuntaviivoja mahdolliselle jatkotutkimukselle.

4.1. Reaaliaikaisuuden asettamat rajoitteet

Alkuperäinen VideoAnalyzerin vaatimusmäärittelyssä asetettu laitteistovaatimus oli Pentium 75 MHz. Testikoneena käytettiin 350 MHz:n Pentium II konetta, jossa oli 128 megatavua muistia ja käyttöjärjestelmänä Windows NT 4.0 Workstation. Tällä koneella videota pystyttiin prosessoimaan klusteritaulukkomallin toisella versiolla ja nykyisillä tietorakenteilla reaaliajassa (AVI-tiedostosta) noin 10 ruudun sekuntinopeudella. Alkuperäinen laitevaatimus oli siis todella vaatimaton nykyiselle reaaliaikaisuuteen pyrkivälle toteutukselle.

Analyysiin on olemassa kaksi vaihtoehtoa. Video prosessoidaan joko reaaliajassa videoilta digitointikortin avulla tai jälkikäteen AVI-tiedostosta. Reaaliaikainen järjestelmä säästää aikaa, kun videota ei tarvitse ensin digitoida tiedostoon. Toisaalta se vaatii käytettävältä laitteistolta huomattavasti enemmän tehoa kuin AVI-tiedoston prosessointi. AVI-tiedoston prosessointi on hidasta, koska video pitää ensin digitoida

tiedostoon. Etuna on se, että käsittelyn ei tarvitse olla reaaliaikaista, jolloin käytettävät algoritmit voivat olla paljon monimutkaisempia. Tapojen erot voivat tuntua marginaalisilta, sillä nykyiset koneet ovat niin tehokkaita, että reaaliaikaisen analyysin ei pitäisi tuottaa minkäänlaisia ongelmia. Pitää kuitenkin huomata, että jos käsiteltävän kuvan kokoa kasvatetaan kaksinkertaiseksi, käsiteltävän datan määrä nelinkertaistuu. Datan valtavan määrän vuoksi tietorakenteiden ja algoritmien optimoinnilla on erittäin suuri merkitys analyysin tehokkuuteen. Seuraavassa alaluvussa kerron, millaisia parannuksia eri toteutuksilla on saatu aikaan.

Ohjelman suorituskyky luultavasti paranisi, mikäli kuva tulisi suoraan videoilta digitointikortin kautta. AVI-tiedostojen käsittelyyn käytetty kirjasto tuntui nimittäin olevan todella raskas varsinkin muistin käytön osalta. Nykyisillä koneilla levy-I/O:n vähentyminen tuskin vaikuttaisi merkittävästi tulokseen.

4.2. Kokeiltujen tekniikoiden arviointia

Kokeilluista tekniikoista on erotettavissa kaksi erilaista lähestymistapaa. Kehysmallin sisäkkäiset raamit seuraavat käyttäjän määrittelemiä hahmoja, ja muut klusterointiin perustuvat tekniikat etsivät kentältä mielivaltaisia hahmoja. Koska tarkoituksena on ollut kehittää automaattinen järjestelmä hahmojen seurantaan, ovat klusterointimallit mielekkäämpi valinta. Kehysmallissa käyttäjän pitää osoittaa kentälle ilmestyvät uudet pelaajat ja algoritmin kadottamat pelaajat analyysin edetessä. Vaikka pelaajat pitää klusterimalleissakin nimetä manuaalisesti, se voidaan tehdä yhdellä kertaa automaattisen analyysin jälkeen.

Klusterointitekniikoiden tarkkuuksien vertailu on ollut hyvin epäformaalia. Kehysmalli ei sovellu usean hahmon seurantaan, joten sitä ei ole testattu samoilla jalkapallovideoilla kuin muita tekniikoita. Klusterimallista luovuttiin niin pian, että siitä ei jäänyt jälkeen vertailukelpoisia testituloksia. Ainoastaan taulukkomallien tuloksia on voitu vertailla hieman tarkemmin – kuitenkin vain silmämääräisesti. Tarkempaa vertailua vaikeuttaa se, että jo muutaman pikselin heitot kentän kiintopisteiden asetuksessa vaikuttavat analyysin lopputulokseen. Ilman taulukkomallin toisen version polkujen suodattamista ja heuristiikka-algoritmia taulukkomallien tulokset ovat lähes identtiset. Suodattamisen merkitystä demonstroitiin jo aikaisemmin kuvassa 3.7.

Seuraavalla sivulla olevasta taulukosta 4.1 käy ilmi eri tekniikoiden ominaisuuksia. Klusterimalli ja klusteritaulukkomallin ensimmäinen versio on laitettu samaan sarakkeeseen, sillä ne ovat pitkälti samankaltaisia. Tekniikoita esitellään tarkemmin seuraavissa alaluvuissa.

TESTATTAVA OMINAISUUS	KEHYSMALLI	KLUSTERIMALLI / TAULUKKOMALLI v1	TAULUKKOMALLI v2
salliiko useita hahmoja	sallii etäällä olevia hahmoja	sallii, mutta ei kosketusta	sallii, mutta ei kosketusta
objektien lukumäärän vaikutus laskentaan	vaikuttaa suuresti	vaikuttaa jonkin verran	ei vaikutusta
nopean liikkeen vaikutus	hahmo karkaa kehyksistä	objektit saattavat sekaantua	objektit saattavat sekaantua
salliiko hahmojen pyörimisen	kyllä	kyllä	kyllä
vaikein tilanne	päällekkäiset hahmot	päällekkäiset hahmot	päällekkäiset hahmot
tunnistaako uusia hahmoja	ei	kyllä	kyllä
käyttääkö väri-informaatiota	ei	ei	kehitteillä

Taulukko 4.1. Tekniikoiden ominaisuuksien vertailua (mukaeltu Siermalalta [1999]).

Käytettyjen tekniikoiden tehokkuuden vertailu on myös ollut varsin epäformaalia. Tarkempia vertailutuloksia on ainoastaan taulukkomallin ensimmäisestä ja toisesta versiosta. Tulokset on esitetty taulukossa 4.2. Analyysissä käytetty video oli kooltaan 23,4 Mt ja pituudeltaan 37 sekuntia. Video oli tallennettu kahden ruudun sekuntinopeudella, mikä on erittäin alhainen nopeus. Taulukosta voidaan havaita, että tekniikoiden tehokkuuksissa on merkittäviä eroja. Huomattavinta on se, että taulukkomallin toisella versiolla päästään reaaliaikaisuustavoitteeseen. Muistin tarpeesta ei vertailussa käytetyn videon kanssa muodostu kummankaan tekniikan tapauksessa kynnyskysymystä, sillä analyysiin käytetty video oli erittäin lyhyt. Mikäli käytetty video olisi ollut kokonaisen puoliajan pituinen, muistin tarve olisi taulukkomallin ensimmäisellä versiolla ollut noin 152 Mt ja toisella versiolla noin 36 Mt. Mikäli video olisi ollut tallennettu standardilla 25 ruudun sekuntinopeudella, olisivat muistin kulutus ja analyysiin käytetty aika olleet 12,5 kertaisia (1900 Mt / 450 Mt).

Tekniikka	Muistin tarve (Kt)	Aika (s)
Taulukkomalli v1	2132	66
Taulukkomalli v2	504	9

Taulukko 4.2. Tekniikoiden tehokkuuksien vertailua.

Kaikki tekniikat sallivat useiden hahmojen seuraamisen, mutta eivät niiden päällekkäisyyttä. Kehysmalli on herkin – se voi sekoittaa hahmot, vaikka ne eivät edes

koskettaisi toisiaan. Taulukkomallit häiriintyvät hahmojen päällekkäisyydestä vain hetkellisesti. Ne sivuuttavat päällekkäisyydet ja jatkavat hahmojen seuraamista, kun polut ovat jälleen ristiriidattomia. Hahmojen lukumäärä vaikuttaa laskennallisesti eniten kehysmalliin. Toisaalta se on laskennallisesti nopein tekniikka ja on joka tapauksessa nopeampi kuin muut. Klusterimallissa ja klusteritaulukkomallin ensimmäisessä versiossa käytetyt tietorakenteet ovat melko tehottomia ja kuluttavat paljon muistia, joten objektien määrä vaikuttaa laskentaan jonkin verran. Klusteritaulukkomallin toisessa versiossa tietorakenteet on mietitty uudelleen ja algoritmeja optimoitu, joten objektien määrä ei vaikuta laskentaan.

Nopea liike häiritsee kaikkia tekniikoita. Kehysmallissa objektit voivat karata kehyksistä ja muissa tekniikoissa polunetsintäalgoritmi saattaa tehdä virhetulkintoja. Kun pelaaja pyörii kentällä, tämän hahmon väripisteet liikkuvat. Tekniikat tunnistavat vain värin muutoksen, joten hahmon pyöriminen ei vaikeuta seurantaa. Pyöriminen voi jopa helpottaa ongelmia, sillä pyöriminen tunnistetaan, vaikka hahmo olisi muutoin paikallaan [Siermala, 1999]. Kehysmalli ja klusterimalli ainakin hyötyvät tästä, sillä ne tunnistavat vain liikettä.

Vaikeinta kaikille tekniikoille ovat päällekkäiset hahmot. Tämä on yleensä ottaenkin suurin ongelma hahmontunnistuksessa. Kehysmalli on kokeilluista tekniikoista ainoa, jolle seurattavat hahmot pitää näyttää käsin. Muut tekniikat etsivät hahmoja automaattisesti. Hahmojen väri-informaatiota ei analyysissä ole hyödynnetty. Klusteritaulukkomallin toinen versio kyllä tallentaa myös pelaajien väri-informaatiota tiedostoon, mutta informaatiota ei vielä käytetä hyväksi missään. Väri-informaation hyödyntämiseen liittyy aika paljon ongelmia, jotka pitäisi vielä ratkaista (ks. luku 4.3).

Kehysmalli

Erotuskuvat on tehokas tapa tunnistaa liikettä kuvissa, ja ne soveltuvat hyvin käytettäväksi sisäkkäisten raamien kanssa, sillä raamien käyttö pienentää prosessoitavaa aluetta huomattavasti [Siermala *et al.*, 2000]. Kehysmalli seuraa sille näytettyjä hahmoja. Se ei itse etsi uusia.

Kehysmallin sisäkkäisten raamien ongelmana on se, että ne eivät pysty käsittelemään muuttuvaa taustaa ja liikkumattomia objekteja. Tässä tutkimuksessa ensimmäinen ongelma ei ole haitannut, sillä tausta pysyy koko ajan paikallaan. Liikkumattomien hahmojen ongelma onkin suurempi. Vaikka pelaajat liikkuvat lähes koko ajan, on myös tilanteita, joissa pelaajat seisovat paikallaan.

Sisäkkäiset raamit osaavat seurata yhtä hahmoa varsin hyvin. Harvoin kuitenkin voidaan tarkastella vain yhtä hahmoa. Usean hahmon tapauksessa raamit ”tarttuvat”

törmätessään toisiinsa kiinni, eli hahmojen liikkeet sulautuvat yhdeksi differenssimatriisin liikealueeksi. Sama ongelma esiintyy myös, jos raamien ohi liikkuu alustamaton hahmo. Toisin sanoen, on vaikea löytää sääntöjä, joilla raamien tarttuminen toisiinsa voitaisiin ehkäistä.

Kehysmalli on varsin kömpelö usean hahmon tapauksessa myös siksi, että kehykset ulottuvat hahmojen ulkopuolelle; toiset hahmot saattavat siis sekoittaa kehyksien avulla tapahtuvaa päättelyä, vaikkeivät edes törmää seurattavaan hahmoon. Ratkaisuna yritettiin rakentaa kehyksiin ominaisuuksia, joilla ne tunnistavat toisen hamon kehykset ja sulkevat nämä alueet päättelyn yhteydessä. Menetelmä ei kuitenkaan toiminut, koska iso hahmo saattoi lamaannuttaa pienen hahmon koko päättelyalueen ja pieni hahmo hävitettiin. Lähelle ajautuvat hahmot voidaan myös yhdistää yhdeksi kehykseksi, jolloin kehyksen äkillisen pinta-alan muutoksen avulla voidaan tarkastella hahmojen eroamista. Tämä menetelmä jäi kuitenkin toteuttamatta. [Siermala, 1999].

Ongelmia syntyy myös siinä tapauksessa, että seurattava hahmo liikkuu nopeasti – hahmo voi olla joko fyysisesti nopea, tai videon tallennusnopeus voi olla alhainen. Pahimmassa tapauksessa seurattava hahmo liikkuu kahden ruudun välillä useiden itsensä kokoisten alueiden verran. Koska kehysten täytyy pystyä tunnistamaan hahmon uuden sijainnin kauimmainen reuna, kehysten on haravoitava iso alue hahmon ympäriltä. Tämä puolestaan aiheuttaa sen, että kehykset menevät helpommin päällekkäin ja kehykset kadottavat hahmon. [Siermala, 1999]

Kehysmalli osaa mukautua seurattavan hahmon koon, liikkeen ja asennon muutoksiin, kunhan eri hahmojen kehykset eivät törmää toisiinsa. Kehysmalli sopii hyvin yhden hahmon seuraamiseen tilanteessa, jossa tausta ei liiku. Kehysmalli on laskennallisesti erittäin nopea, sillä differenssimatriisi lasketaan ainostaan raameille. [Siermala, *et al.*, 2000]

Klusterimallit

Klusterimallien merkittävä etu kehysmalliin on se, että seurattavia hahmoja ei tarvitse käsin määritellä, vaan ne löytyvät automaattisesti. Tämä tietysti edellyttää sitä, että koko kuva pitää käydä läpi. Siksi klusterimallit ovatkin laskennallisesti huomattavasti raskaampia kuin kehysmalli.

Klusterimalli kiinnittää vain tietyn säteen etäisyydellä olevat liikepisteet toisiinsa. Tämän vuoksi menetelmä on huomattavasti ongelmattomampi usean hahmon tapauksessa kuin kehysmalli. Kun hahmo liikkuu useita itsensä kokoisia alueita eteenpäin, pitää kehysmallin tunnistaa hahmon uuden sijainnin kauimmainen reuna. Klusterimalleissa riittää, että hahmoa seuraavassa ruudussa lähinnä oleva objekti on

hahmo itse, eikä muita hahmoja ole kosketusetäisyydellä. Läheisten hahmojen liikealueet voivat kuitenkin sulautua yhdeksi alueeksi ja menetelmä kadottaa hahmot. [Siermala, 1999]

Klusterimalli tekee polkupäätökset heti kuvaa analysoitaessa. Valitettavasti kuvissa on paljon turhaa dataa, ja klusterimalli löytää paljon liian lyhyitä ja virheellisiä polkuja [Siermala *et al.*, 2000]. Tämä lisää laskennallista taakkaa. Muutoin liian lyhyistä poluista ei ole haittaa, sillä ne voidaan karsia pois automaattisen analyysin jälkeen.

Klusteritaulukkomalleissa käytetty mallin sovittamiseen perustuva tekniikka toimii paremmin kuin kahden peräkkäisen ruudun erotuskuvat, sillä pelaajat eroavat selkeästi väriltään kentästä ja löytyvät näin ollen myös liikkumattomina.

Klusteritaulukkomallin ensimmäinen versio kerää kaikki klusterit taulukkoon ja tämän jälkeen alkaa polkujen etsiminen. Käyttäjältä tiedustellaan hahmojen (pelaajien) nimiä järjestyksessä pisimmistä poluista lähtien. Näin pystytään tehostamaan automatisoitavissa olevat tehtävät. [Siermala, 1999]

Klusterimalli ja klusteritaulukkomallin ensimmäinen versio käyttävät klustereiden muodostamiseen pistealgoritmia. Kelpuutettujen pikseleiden lukumäärä vaikuttaa suuresti pistealgoritmin suoritukseen. Linkitettyjen listojen käyttö on osasyy tähän. Vielä merkittävämpää on se, että pisteet joudutaan käymään useaan kertaan läpi. Rekursioon perustuvassa klusteritaulukkomallin toisessa versiossa kuvan pisteitä ei tarvitse käydä kuin kerran läpi. Klusteritaulukkomallin toinen versio onkin noin seitsemän kertaa nopeampi kuin ensimmäinen versio. Kun käsiteltävien ruutujen määrää vähennetään vielä esimerkiksi puoleen, on nopeusero jo 14-kertainen. Tietorakenteilla on myös suuri merkitys nopeuden kasvamiselle. Lisäksi tietorakenteet vaikuttavat merkittävästi muistin kulutukseen.

Klusterimallien ongelma on se, että systeemissä on paljon vakioita, joista osa on adaptiivisia. Ohjelman pitää esimerkiksi tietää pikselin maksimietäisyys tiettyyn klusteriin kuulumiselle ja värikomponenttien raja-arvot. Optimaaliset arvot näille vakioille täytyy selvittää kokeellisesti. [Siermala *et al.*, 2000]

4.3. Hahmojen päällekkäisyyden hallinta

Hahmojen päällekkäisyys on ollut tutkimuksen kannalta suurin ongelma. Hahmojen päällekkäisyys aiheuttaa sen, että klusterointialgoritmi ei aina löydä kaikkia hahmoja. Tämän seurauksena polkujenetsintäalgoritmi ei aina osaa muodostaa oikeita polkuja. Ongelmaa on alan kirjallisuudessa yritetty ratkaista muun muassa liikkeiden

ennustamista ja väri-informaatiota hyödyntämällä. Siermala *et al.* [2000] suosittelevat älykkäiden heuristiikkojen käyttöä.

Jalkapalloilijoiden liikkeiden ennustaminen on vaikeaa, koska pelaajat vaihtavat suuntaa erittäin nopeasti ja usein. Kameran sijainnista johtuen systeemi ei erota pelaajien hyppyjä ja kaatumisia normaaleista liikkeistä. Hyppyjen ja kaatumisten ennustaminen on luonnollisesti täysin mahdotonta.

Väri-informaation hyödyntäminen puolestaan on vaikeaa siksi, että seurattavat hahmot ovat VideoAnalyzerin tapauksessa erittäin pieniä. Hahmon (pelaajan) keskimääräinen väri sisältää helposti pääasiassa taustan murtamia sävyjä esimerkiksi digitoinnissa tapahtuneesta interpoloinnista johtuen. Näin kaikki pelaajat ymmärretään väri-informaation perusteella samoiksi. Kuvassa 4.1 on suurennokset kahdesta käsin piirretystä hahmosta. Niiden väritys eroaa selkeästi toisistaan. Kuitenkin kuvan interpolointi samankaltaistaa niitä huomattavasti. Jalkapallovideoiden tapauksessa hahmojen värierot ovat yleensä paljon pienempiä kuin kuvan esimerkissä. Ainakin saman joukkueen pelaajilla on todennäköisesti samanväriset vaatteet. Lisäongelmia tulee, mikäli pelipaidan selkäpuoli on eri värinen kuin etupuoli. Silloin yksittäisen pelaajankin havainnot värin perusteella voivat olla epäkelpoja eri ruutujen välillä. Yksi ratkaisu väri-informaation paikkaansapitävyyden kasvattamiseen on kuvan resoluution kasvattaminen. Resoluutiota rajoittavana tekijänä on reaaliaikaisuuden vaatimus. Ainakaan testilaitteistolla kuvan resoluutiota ei olisi voitu kasvattaa yhtään. Täytyy kuitenkin muistaa, että edes ihminen ei aina kykene erottamaan jalkapallovideosta, kuka kukin on. Yhden kameran systeemillä analyysin tulos on tuskin koskaan täydellinen.

(a) alkuperäinen kuva

(b) Sama kuva puoleen kokoon skaalattuna.

Kuva 4.1. Interpoloinnista johtuva hahmojen väri-informaation samankaltaistuminen.

Polkujen etsinnän yhteydessä kokeilemani jaetun polun heuristiikka aiheutti loputtomia silmukoita ja jäi siksi vähemmälle huomiolle. Sen avulla kuitenkin jotkut päällekkäisyystilanteet saatiin ratkaistua automaattisesti. Hahmojen koon muuttumisen lisäksi apuna olisi voitu käyttää väri-informaatiota. Klusteritaulukkomallin toinen versio etsii hahmoja vertaamalla pisteitä analyysin alussa annettuun kenttäesimerkkiin ja vierekkäisten pikseleiden kontrastierojen perusteella. Malli ei kuitenkaan hyödynnä pelaajien väri-informaatiota.

Hahmon keskimääräisen värin laskemiseen kannattaa ehkä käyttää vain taustasta eroavia pikseleitä. Korkeakontrastiset pisteet ovat monesti varjoja ja ääriviivoja, jotka samankaltaistavat hahmoja kuvassa 4.1 esitetyllä tavalla. Hahmon ”sisukset” säilyttävät yleensä hahmon ominaisvärin. Vaikka resoluutiota voitaisiin kasvattaa äärettömästi ja pelaajista saada tarkkaa väri-informaatiota, hahmot menevät silti päällekkäin. Heuristiikat eivät ole riippuvaisia kuvan laadusta vaan ne hyödyntävät sovellusalueen tietämystä. Tämän vuoksi heuristiikat voivat olla suuri apu hahmojen päällekkäisyyden hallinnassa.

4.4. Johtopäätöksiä ja mietteitä jatkotutkimukseen

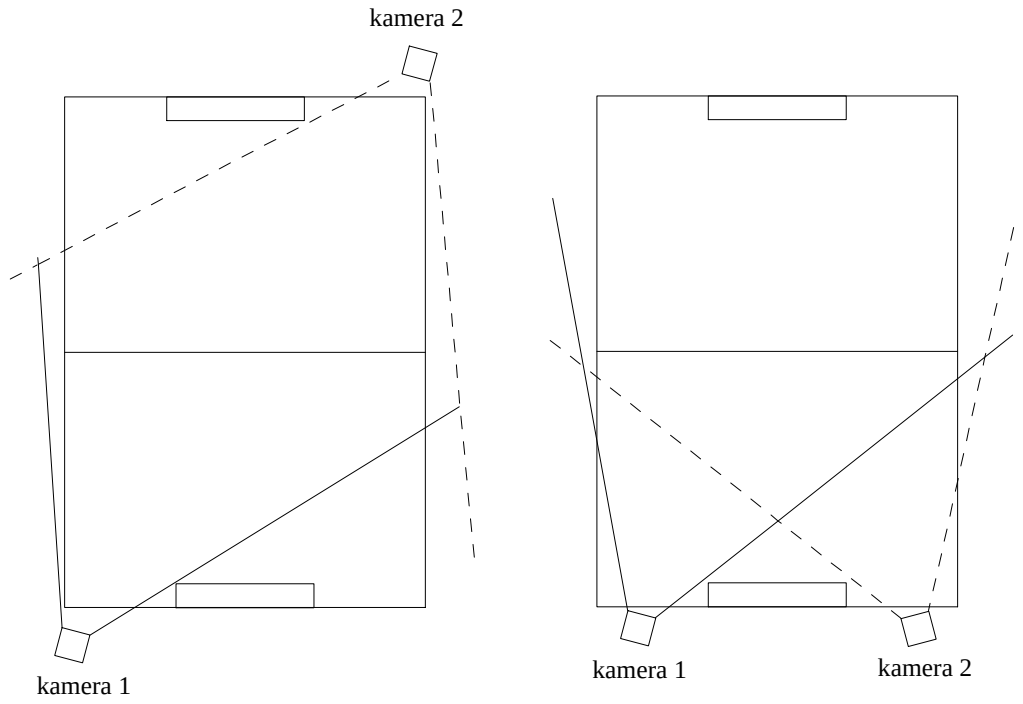
Automaattisella videoanalyysillä on käytännössä mahdotonta saada täydellistä peli-informaatiota. Tämä ei ole mikään ihme, sillä ihmisenkin on joskus mahdotonta erottaa pelaajia toisistaan videokuvasta. Tehtävän tekee erityisen vaikeaksi tilanteet, joissa pelaajat menevät kameraan nähden limittäin. Klusteritaulukkomenetelmän toinen versio toimii kokeilluista tekniikoista parhaiten. Se ei mene sekaisin pelaajien mennessä päällekkäin kuten kehysmalli – se vain sivuuttaa tällaiset tilanteet. Epäselvät tilanteet voidaan käsitellä manuaalisesti automaattisen analyysin jälkeen. Taulukkomallin toinen versio on myös huomattavasti nopeampi ja käyttää vähemmän muistia kuin klusterimalli ja klusteritaulukkomallin ensimmäinen versio.

Hahmojen päällekkäisyyden käsittelyä voidaan helpottaa erilaisilla heuristiikoilla. Useissa tutkimuksissa päällekkäisyystilanteiden selvittämiseen käytetään hahmojen väri-informaatiota. VideoAnalyzerin tapauksessa hahmot ovat kuitenkin niin pieniä, että niiden väri-informaation hyödyntäminen on vaikeaa. Alustavien testien perusteella hahmojen keskimääräiset värit ovat liian samankaltaisia, että niitä olisi mielekästä hyödyntää. Yksi mahdollinen ratkaisu on, että hahmon reunojen värejä ei oteta mukaan keskimääräisen värin laskemiseen.

Hahmojen päällekkäisyyden lisäksi valaistuksen vaihtelut ja varjot aiheuttavat päänvaivaa hahmontunnistusalgoritmeille. Erotuskuvia hyödynnetään kehysmallissa ja klusterimallissa vertailemalla kahta peräkkäistä ruutua. Ongelmana on, että näin tunnistetaan vain liikkeet. Erotuskuvia voitaisiin hyödyntää VideoAnalyzerissä ottamalla vertailukuva tyhjästä kentästä. Näin myös paikallaan olevat hahmot tunnistettaisiin. Erotuskuvat ovat myös herkkiä valaistuksen vaihteluille, mutta ne toimivat paremmin tilanteissa, joissa tietyt osat kentästä ovat huomattavasti eri värisiä kuin muut. Jos kentällä näkyy esimerkiksi varjo läheisestä talosta, se luultavasti tunnistetaan klusteritaulukkomalleilla hahmoksi, mikäli tätä varjoa ei ole sisällytetty kentästä annettavaan esimerkkiin. Jos taas varjo on mukana esimerkissä, taustan

värihajonnasta saattaa tulla niin laaja, että klusterointialgoritmi ei erota pelaajia kentästä.

Analyysin tarkkuus laskee huomattavasti, kun kameran etäisyys pelaajiin ylittää puoli kenttää. Yksi mahdollisuus olisi käyttää kahta kentän eri päissä sijaitsevaa kameraa ja yhdistää näiden havainnot. Toinen mahdollisuus olisi käyttää kahta kentän samassa päässä eri kulmissa sijaitsevaa kameraa. Kameroiden havainnot yhdistämällä hahmojen syvyysetäisyydestä saadaan huomattavasti tarkempaa tietoa kuin yhden kameran systeemissä. Kameroiden sijoittelua on havainnollistettu kuvassa 4.2 VideoAnalyzeriin olisi melko helppo lisätä tuki useammalle videolähteelle. Useamman videolähteen käyttö ei kuitenkaan oikein sovi reaaliaikaiseen systeemiin, sillä kahden videokuvan yhtäaikainen digitointi vaatisi erikoislaitteistoa tai useampaa verkotettua tietokonetta.



(a) Kaksi kameraa kentän eri päissä.

(b) Kaksi kameraa kentän samassa päässä.

Kuva 4.2. Kahden kameran sijoittelumahdollisuuksia.

Analyysin tarkkuutta voidaan lisäksi parantaa kuvan resoluutiota kasvattamalla. Tämä on kuitenkin ongelmallista, sillä kuvan tarkkuuden kaksinkertaistaminen nelinkertaistaa käsiteltävän datan määrän. Videokuvan parillisten ja parittomien vaakarivien välisestä vaihe-erosta johtuen kuvan havaittua tarkkuutta voidaan parantaa käyttämällä pelkästään parillisia tai parittomia vaakarivejä. Koska jalkapallokenttä on pidempi pystysuunnassa kuin vaakasuunnassa, on pystysuuntainen tarkkuus hahmojen todellisen sijainnin selvittämisen kannalta tärkeämpää. Kuvan vaihe-eroista johtuen videon pystysuuntainen tarkkuus on käytännössä vain puolet todellisesta tarkkuudesta, eli

kuvan resoluutio on PAL-järjestelmässä maksimissaan käytännössä 720 x 288. Pitäisikö kameran siis kenties olla poikittain? Uhkana on, että vaakasuuntainen sektori saattaa tulla liian pieneksi. Tähän kysymykseen olisi syytä perehtyä tarkemmin.

VideoAnalyzerissä analyysi tapahtui pitkään bittikarttojen koordinaateilla. Tämän vuoksi perspektiivin vaikutusta ei otettu huomioon. Vasta klusteritaulukkomallin toisen version polkujenetsinnän heuristiikka-algoritmi poistaa perspektiivistä johtuvia virhehavaintoja. Perspektiivin vaikutuksen huomioiminen muissakin analyysin vaiheissa on yksi jatkokehityksen aihe. Esimerkiksi klustereiden minimi- ja maksimikokovaatimusten tulisi myös olla suhteessa pelaajan sijaintiin kentällä.

Tällä hetkellä VideoAnalyzerissä on vaatimuksena, että kameran tulee sijaita kentän vasemmassa laidassa. Kameran sijaintivaatimus voitaisiin poistaa melko helposti. Kameran sijainnin sanelee yhden kameran systeemissä kuitenkin ennemminkin se, että suurin osa kentästä saadaan parhaiten mahtumaan kuvaan, kun kamera sijoitetaan kentän vasempaan tai oikeaan laitaan.

Lähteet

[Agbinya and Rees, 1999] Johnson I Agbinya, David Rees. Multi-Object Tracking in Video. *Real-Time Imaging 5, 295-304*, 1999.

[Camus, 1997] Ted Camus. Real-Time Quantized Optical Flow. *Real-Time Imaging 3*, 71-86, 1997.

[Chang and Lee, 1997] Chueh-Wei Chang, Suh-Yin Lee. A Video Information System for Sport Motion Analysis. *Journal of Visual Languages and Computing 8*, 265-287, 1997.

[Choi *et al.*, 1997] Sunghoon Choi, Yongduek Seo, Hyunwoo Kim, Ki-Sang Hong. Where are the ball and players? Soccer Game Analysis with Color-based Tracking and Image Mosaick. Dept. of EE, Pohang University of Science and Technology, Korea, 1997.

[Gunn, 1996] Steve R. Gunn. Dual Active Contour Models for Image Feature Extraction. Faculty of Engineering and Applied Science, Department of Electronics and Computer Science, 1996.

[Hawkins, 1970] J. K. Hawkins. Image Processing Principles and Techniques. Julius T. Tou (ed.). *Advances in Information Systems Science*, vol 3, Plenum Press, New York – London, 117-118; 121-127, 1970.

[Jain *et al.*, 1999] A.K. Jain, M.N. Murty, P.J. Flynn. Data Clustering: A Review. *ACM Computing Surveys, 31 (3)*, September 1999.

[Kehtarnavaz and Rajkotwala, 1997] N. Kehtarnavaz, F. Rajkotwala. Real-Time Vision-based Detection of Waiting Pedestrians. *Real-Time Imaging 3*, 433-440, 1997.

[Kim *et al.*, 1997] Heung-Nam Kim, Mary Jane Irwin, Robert Michael Owens. Motion Analysis on the Micro Grained Array Processor. *Real-Time Imaging 3*, 101-110, 1997.

[Kovalevsky, 1970] V.A. Kovalevsky. Pattern recognition: heuristics or science? Julius T. Tou (ed.). *Advances in Information Systems Science*, vol 3, Plenum Press, New York – London, 2-3; 18, 1970.

[Li and Wang, 1999] Zheng Li, Han Wang. Real-Time 3D Motion Tracking with Known Geometric Models. *Real-Time Imaging 5*, 167-187, 1999.

[Luhtanen, 1999] *Suullinen tiedonanto*. Pekka Luhtanen, pluhta@kihu.jyu.fi, Jyväskylä, 1999.

[MAOL, 1992] MAOL-taulukot. Matemaattisten Aineiden Opettajien Liitto ry. Kustannusosakeyhtiö Otavan painolaitokset, Keuruu, 1992.

[Marchant *et al.*, 1998] John A. Marchant, Robin D. Tillett, Renaud Brivot. Real-Time Segmentation of Plants and Weeds. *Real-Time Imaging 4*, 243-253, 1998.

[Pla and Marchant, 1997] Filiberto Pla, John A. Marchant. Matching Feature Points in Image Sequences through a Region-Based Method. *Computer Vision and Image Understanding, 66 (3)*, 271-285, 1997.

[Rosin, 1997] Paul L. Rosin. Edges: saliency measures and automatic thresholding. *Machine Vision and Applcations 9*, 139-159, 1997.

[Siermala, 1999] Markku Siermala. Jalkapallo-pelin liikeanalyysi. Raportti B-1999-4. Seppo Visala (toim.). Pieniä ATK-alan tutkimuksia, Tietojenkäsittelyopin laitos, Tampereen yliopisto, 1999.

[Siermala *et al.*, 2000] Markku Siermala, Mikko Kankainen, Jyrki Nummenmaa. Tracing footballers' movements from video. Report A-2000-19. Department of Computer and Information Sciences, University of Tampere, 2000.

[Siyal and Fathy, 1999] M.Y. Siyal, M. Fathy. Image Processing Techniques For Real-Time Qualitative Road Traffic Data Analysis. *Real-Time Imaging 5*, 271-278, 1999.

[Smith, 1998] S. M. Smith. ASSET-2: Real-Time Motion Segmentation and Object Tracking. *Real-Time Imaging 4*, 21-40, 1998.

[Tan *et al.*, 1999] Teewoon Tan, Ling Guan, John Burne. A Real-time Image Analysis System for Computer-Assisted Diagnosis of Neurological Disorders. *Real-Time Imaging 5*, 253-269, 1999.

[Tohka, 1998] Jussi Tohka. Madoista. Julkaisematon raportti, 1998.

[Tohka, 2001] Jussi Tohka. Note on Connections between Active Contours and Rayleigh Quotients. Proceedings of 2001 Finnish Signal Processing Symposium, Espoo, 2001. http://wooster.hut.fi/publications/finsig01/note_on_connections.pdf. Luettu 20.3.2002.

[Watanabe, 1970] Satosi Watanabe. Feature compression. Julius T. Tou (ed.). *Advances in Information Systems Science*, vol 3, Plenum Press, New York – London, 63-64, 1970.

[Yeasin and Chaudhuri, 2000] Mohammed Yeasin, Subhasis Chaudhuri. Development of an Automated Image Processing System for Kinematic Analysis of Human Gait. *Real-Time Imaging 6*, 55-67, 2000.